МОНИТОР

журнал для программистов-практиков

7-8.93

ISSN 0869-3889

Интервальные методы в графике

ТЕХНОЛОГИИ

Сжатие по Хаффмену

ЯЗЫКИ

Pascal вчера, сегодня, завтра

TopSpeed Pascal

ГРАФИКА

Масштабирование растровых картинок

19200 BPS
Сертификат Минсвязи РФ
№ ОС-1-ТМ-41

БЫСТРО. ПРОСТО. НАДЕЖНО.

Ваши потребности уже выросли за пределы нынешнего модема. Вам нужен факс—модем, который действительно отвечает современным стандартам. В самом деле выбор предельно простой. Только ZyXEL предлагает Вам универсальные, высокоскоростные факс—модемы с голосовыми возможностями: успешно проверенные на российских телефонных линиях, с квалифицированной поддержкой, по цене, которая не пугает. Ведь телефонное время тоже стоит денег...

Официальный дистрибьютер
А/О МКЦ Вариант
117279 Москва, ул. Островитянова, 37а
Тел. 932—7601, 420—2519
Факс 932—7410, 420—5311

Круглосуточно:
Голос/Факс демо 932—8510
Информация
max@mechta.msk.su2:5020/22@fidonet
WHITE BEAR BBS 932-8465!

Многолетнему господству процессорной архитектуры Intel в мире персональных компьютеров брошен вызов. 32-битовый RISC-процессор PowerPC, разработанный объединенными усилиями фирм IBM, Apple и Motorola, станет базовым процессором для новых поколений персональных компьютеров.

В настоящее время уже доступна 66 МГц версия PowerPC — MPC601. Ее выпускает фирма IBM; цена — 450 долларов — примерно вдвое дешевле, чем Pentium; производительность на некоторых операциях в 5 раз выше, чем у Pentium.

К концу 1993 года IBM собирается выпустить на базе MPC601 настольный компьютер, портативный компьютер и рабочую станцию RS/6000. Фирма Apple намерена выпустить первый Macintosh с MPC601 в первой половине 1994 года.

На PowerPC уже реализованы или будут выпущены в 1994 году все основные операционные системы: Apple System 7, OS/2, IBM AIX, Sun Microsystem Solaris, PowerOpen Environment, Pink (проект Taligent фирм Apple и IBM), Windows NT, Novell Netware, Unix System V.

Предполагается, что все богатство Intel совместимых программ будет исполняться на PowerPC с помощью специально разработанных программ-эмуляторов. При эмуляции производительность снижается по сравнению с производительностью программ, написанных для PowerPC, примерно до уровня Intel 486.

Проектировщики компьютерных сисем на базе PowerPC смогут использовать самые разные архитектурные решения. Так, например, Motorola выпускает наборы кристаллов, позволяющие реализовать в PowerPC системах архитектуры EISA, ISA, PCI и VME.

PowerPC будет применяться не только в персональных компьютерах. IBM намерена строить на базе PowerPC весь спектр своих компьютеров: от портативных до многопроцессорных больших ЭВМ.

В 1994 году ожидается появление еще трех моделей PowerPC: MPC603, MPC604 и MPC620. Все они будут выпущены фирмой Motorola. MPC603 будет отличаться низким потреблением энергии и предназначен для портативных компьютеров. MPC604 с повышенной степенью параллелизма и конвейеризации, 32-разрядной шиной адреса и 64-разрядной шиной данных — для настольных компьютеров следующего поколения. MPC620 с 64-разрядными шинами адреса и данных — для рабочих станций и серверов.

Сбудутся ли эти планы и сможет ли в действительности PowerPC составить конкуренцию Pentium и последующим процессорам Intel — покажет время. Подробную статью про архитектуру процессора PowerPC вы сможете прочитать в одном из ближайших номеров «Монитора».

В заключение — информация для наших подписчиков. В 1994 году выйдут 8 номеров «Монитора» и 4 номера «Монитор-Аспекта». Подписной купон на «Монитор» вы найдете на стр. 82; подписной купон на «Монитор-Аспект» — на стр. 55.

РЕКЛАМА В НОМЕРЕ:

Autodesk RF	*2-я вкл.*	ИНСАЙТ	*25*
Elit-Soft	*71*	Радом	*96*
INTERLINK	*25*	Рестэк	*3-я обл.*
LVS	*1-я вкл.*	Софт-Москва	*4-я вкл.*
Symantec	*20*	Стиплер	*33*
Вариант	*2-я обл.*	Центр	*4-я обл.*
Грант	*3-я вкл.*	молодежных	
ДЕМОС	*25, 33, 41*	исследований	

Ответственность за информацию, приведенную в рекламных материалах, несет рекламодатель.

Журнал для программистов-практиков

Редколлегия:
Д.Э.Бройтман (главный редактор)
В.В.Водолазский
В.С.Головин
Н.Н.Косарев
А.М.Миляков (ответственный секретарь)
Н.А.Никитина (отдел рекламы)
Е.В.Равич
Л.И.Турок
И.В.Ульянцева

Графическое бюро:
С.В.Макельский (главный художник)
Г.Л.Карчикян (компьютерная верстка)

Отдел распространения:
Р.А.Болдырев
Е.В.Гун
В.Ю.Томская-Ефремова

Дирекция
А.В.Орехов (издательский директор)
А.Д.Гассельблат (коммерческий директор)

Представительство в Европе (Германия)
Marina Guetmanskaia
GLEIWITZER STR 5
67659 KAISERSLAUTERN
Tel. (0631) 7-49-05

В оформлении номера использованы рисунки
А.С.Замулы

Подписано в печать с оригинал-макета 11.11.93
Формат 60x84/8.
Тираж 30 000 экз.
Печать офсетная. Печ. л. 12,0
Цена договорная
Зак. 4543
Отпечатано в ИПК «Московская правда»

Учредитель:
Научно-техническое общество
Софт·Москва

Журнал зарегистрирован в Министерстве печати Российской Федерации. Регистрационный номер 1971

Для писем:	117419 Москва, а/я 765
Телефон	(095) 238-99-33
Факс	(095) 238-32-78
Отдел рекламы	(095) 903-46-26
Для справок	(095) 943-25-53
Телекс	613322 GRANT SU
BBS	(095) 135-14-39 ARSORBIS

© "Монитор", 1993

Содержание

Письма — 4

Загрузка экранных шрифтов
Размышления о «Цивилизации»
Редактор графических курсоров мыши
Рекордов хотите? Их есть у меня
Письмо сердитого читателя
Ответ сердитому читателю

Технологии программирования

А. Морозова. Отечественные CASE-средства разработки проектов информационных систем — 8

Даются основы CASE-подхода как нового направления программной инженерии. Приводится краткое описание десяти CASE-средств, созданных в России в последние годы.

Д. Мастрюков. Алгоритмы сжатия информации. Часть 1. Сжатие по Хаффмену — 14

В статье описывается алгоритм адаптивного кодирования сжатия Хаффмена. Приводится листинг программы на языке С, реализующей этот алгоритм.

Языки программирования

А. Штрик. Языки программирования четвертого поколения. Часть 2 — 26

Во второй части статьи рассматриваются характеристики тринадцати конкретных языков четвертого поколения.

М. Трофимов. Pascal сегодня, вчера, завтра — 34

Аналитический обзор: история и сегодняшнее состояние языка Pascal, относящиеся к Pascal стандарты, языки-конкуренты Pascal, обзор литературы.

А. Тихонов. TopSpeed Pascal — 42

В статье рассмотрена система программирования TopSpeed версии 3.1 фирмы Clarion Software, обладающая многими уникальными особенностями. Анализируются особенности входного языка TopSpeed Pascal, совместимость его с Turbo Pascal, языками семейства TopSpeed и другими системами программирования.

М. Трофимов. Язык программирования «Эллочка» — 56

В порядке отклика на триады Очкова предложен язык программирования «Эллочка».

Выставки

На выставку в северную столицу! — 32

Когда кругом в дыму снаряды рвутся... — 53

MS BASIC

А. Колесов. Профессиональная работа в среде Microsoft BASIC. Часть 3. Когда программы становятся большими — 60

В этом разделе статьи рассматриваются методы оптимизации размеров программ при использовании средств программной обработки ошибок, а также вопросы сокращения времени создания исполняемых модулей.

Содержание

Игры

А. Стариков. «Hero Quest II» — игра для настоящих героев! — 68

Руководствуясь нашими советами, Вы станете одним из них.

К. Бакай. Там на неведомых дорожках... — 72

Методические рекомендации поклонникам игры Goblins.

DOS под микроскопом

Б. Горбунов. Откуда ты, моя программа? — 75

В статье описывается алгоритм определения маршрута доступа к файлу, из которого загружена программа; приводится программа на языке Turbo Pascal.

Мир графики

С. Курковский. Интервальные методы в компьютерной графике — 76

В статье описаны оригинальные алгоритмы для построения графиков, линий уровня и поверхностей различных функций. Приведенные алгоритмы основаны на применении интервальной арифметики, которой посвящен один из разделов статьи. Большое внимание уделяется практической реализации этих методов на языке C++.

А. Крупник. Как масштабировать растровые картинки — 86

Рассматривается метод масштабирования растровых картинок, заключающийся в дублировании или исключении строк и пикселов исходной картинки. Приводится подпрограмма на ассемблере, которая осуществляет масштабирование растровых картинок, записаных в формате .PUT.

Книги

А. Арапов. Процессоров стройные ряды... — 90

Дайджест книги Джерри Л. Поттера «Ассоциативные вычисления. Парадигма программирования для компьютеров с массовым параллелизмом.»

Защита программ

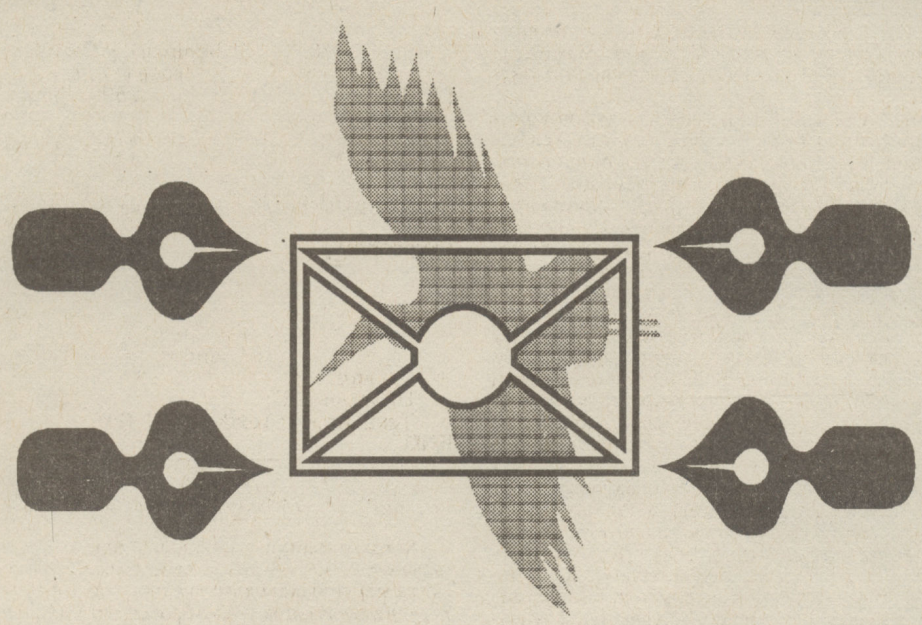
Д. Мухин. Удаление защиты PC-VAULT — 92

В статье рассматривается проблема снятия защиты жесткого диска PC-VAULT. Представлен листинг программы на языке ассемблера, позволяющей удалить данную защиту.

Компьютер в разрезе

С. Дорошев. Если у Вас проблемы с принтером CM6337 — 95

Приведена информация, позволяющая решить проблемы, связанные с эксплуатацией принтера CM6337.



Уважаемые читатели!
До сих пор в этой рубрике мы печатали ваши письма со своим комментарием. Однако увеличение популярности нашего журнала ведет к резко возрастающему объему переписки. Поэтому у нас наконец-то появилась возможность организовать диалог читателей на страницах журнала. В этот раз мы не станем занимать драгоценное место, пусть читатели и авторы, которые тоже читают наш журнал, поделят его между собой сами.

Редакция

Загрузка экранных шрифтов

Уважаемый г-н редактор!

Загрузить русский шрифт в IBM совместимый ПК можно в нерезидентном режиме, что сэкономит около 10 Кбайтов оперативной памяти. Делается это следующим образом:

ah = 11h
al = 00h
es:bp = указатель на таблицу символов (загрузите файл шрифта в буфер и присвойте es:bp указатель на него),
cx = количество символов для загрузки (сколько символов Вы хотите изменить),
dx = код первого изменяемого символа (0-FFh),
bl = font block для загрузки (как правило, равен 0),
bh = высота матрицы (14 для EGA и 16 для VGA).

После этого можно вызывать прерывание INT 10h.

Не забудьте только, что после каждой смены видеорежима Вам придется загружать шрифт заново и что потребность в драйвере клавиатуры по-прежнему сохраняется.

В качестве готового файла с экранными шрифтами можно использовать любой входящий в состав популярной программы **EVAFONT**.

Кстати, не видел ни одного приложения, в котором использовался бы экранный шрифт из 512 символов. Впрочем, скоро мы получим CLIPPER 6.0, который будет работать под управлением Windows, и наконец-то расстанемся навсегда с текстовым интерфейсом.

Ст. лейтенант милиции А. Хохряков, г. Москва ■

Размышления о «Цивилизации»

Здравствуйте, редакция журнала «Монитор»!

Я не являюсь подписчиком вашего журнала, но теперь намерен стать таковым. Недавно я купил в магазине несколько экземпляров «Монитора» за 1992 год. В номере 6-7 была статья «Постижение истории» об игре «Цивилизация». Может быть, для вас эта тема уже не представляет интереса, но в любом случае мне просто хотелось бы подытожить свои знания об этой игре. Это письмо не может претендовать на то, чтобы быть стратегическим исследованием, и содержит лишь некоторые мои советы тем, кто играет в эту интересную и захватывающую игру (я сам сиживал за ней ночами).

Во-первых, я не могу согласиться с некоторыми пунктами самой статьи. Там сказано, что желательно поддерживать мир с соседними цивилизациями, но в этой игре невозможно равноправное существование цивилизаций. Мир заключается только с тем, чтобы выиграть время для наращивания военной мощи, или для того, чтобы войска заняли выигрышную позицию. «Цивилизация» — это прежде всего военная игра.

Далее в статье говорится о дипломатах и торговых караванах. Я могу сказать, что караваны не приносят в казну серьезного дохода, деньги для содержания городских учреждений приходится добывать другими способами. Первый способ — это вымогательство денег у более слабых цивилизаций, а второй, более надежный, состоит в том, чтобы выделять некоторые города для постройки хозяйственных объектов с целью их немедленной продажи.

Что касается дипломатов, то, кроме перечисленных в статье функций, они могут выполнять еще одну — проводить войска по вражеской территории. Как известно, только дипломаты могут передвигаться по чужим дорогам, ирригациям, шахтам (если не объявлено состояние войны — **прим. ред.**). Для того чтобы подвести войска к городу противника, следует передвигать их вместе с дипломатом (на одной клетке).

К вопросу политологии: только две формы правления являются выигрышными — деспотизм и монархия, остальные не дают игроку полного контроля ситуации и ведут к гибели.

Во-вторых, я не согласен с вашим планом, приведенным в конце статьи. После создания первых городов следует не заключать с соседями мирные договоры, а развивать науки, позволяющие создавать боевые единицы (например, колесо) и захватывать города противников, пока они не окрепли. Желательно полностью очистить свой континент от других государств. В 20-м веке, если игрока не интересует, сколько рейтинговых очков он получит, я бы посоветовал широко использовать ядерное оружие против наиболее сильных конкурентов. Кстати, мой рейтинговый рекорд — 56% на уровне PRINCE.

И последнее. Космический корабль можно запускать, как только появляется слово LAUNCH. Даже если говорится, что он имеет 0 шансов на успех, корабль обязательно прилетит.

P.S. Мое мнение о журнале: «Монитор» должен быть журналом только для программистов, журналов для пользователей хватает и без него. Больше печатайте ассемблерного кода, меньше C и Pascal, потому что на ассемблере пишут все грамотные программисты.

А. Янович, г. Владивосток ■

Редактор графических курсоров мыши

Уважаемая редакция!

В № 4 вашего журнала за 1993 год прочитала статью А. Аброськина «Библиотека графических курсоров «мыши». Склоняю голову перед трудолюбием автора и проделанной им кропотливой работой, которая действительно способна «свести с ума». Вероятно, многим читателям пригодится приведенная в статье библиотека графиче-

ских курсоров «мыши». Однако у меня никак не вызывает одобрения предлагаемый подход к решению задачи. Человек должен заниматься творческой работой, рутинную же следует поручить машине.

Предлагаю способ, позволяющий быстро создать любое количество пиктограмм графического курсора «мыши», не «сходя с ума» и не «потея». Приведенная мной программа чрезвычайно проста: она включает небольшой экранный редактор, работающий в текстовом режиме монитора, для формирования изображения и процедуру, считывающую информацию из видеопамяти, осуществляющую кодирование и вывод на экран полученных значений. Выводимые числа можно перевести в шестнадцатеричный формат, однако я не стала этого делать, чтобы не загромождать программу. Более того, я считаю, что пользователю вообще излишне знать, как именно кодируется созданный им рисунок. Логичнее всего сразу записывать получаемые результаты на диск и впоследствии при необходимости считывать их в массив, что и было выполнено при тестировании программы. Впрочем, здесь возможны варианты. Мой пример носит демонстрационный характер, и при желании его можно легко модифицировать и расширить по своему усмотрению (например, применить для рисования ту же «мышь»).

Экранный редактор, включенный в программу, предусматривает использование клавиш перемещения курсора, а также клавиш «HOME», «ENTER» и «забой». При нажатии любой символьной клавиши на экран выводится «квадратик» (ASCII 178), при одновременном нажатии символьной клавиши и клавиши «SHIFT» печатается «звездочка» (ASCII 42). При помощи символа «■» формируется часть изображения курсора, кодируемая первыми 32 байтами (16 словами) матрицы и осуществляющая с содержимым экрана логическую операцию «AND», при помощи символа «*» — оставшиеся 32 байта (логическое «XOR»). Фрагменты изображения различаются также цветом (красный и белый).

Чтение изображения и печать полученных кодов производится при нажатии клавиши «F1». Выход из программы — «Esc».

При работе в системе Turbo Pascal 7.0 программу для разработки курсора можно включить в меню «Tools» и вызывать при необходимости, что очень удобно.

С уважением, Н. Васильева, г. Барнаул ■

Листинг

```
Program Mouse_Cursor;
uses CRT;

var n,w: word;
    x,y,k,i : byte;
    m,m1 : char;

procedure Print(sym: char);
                          { Чтение изображения, кодирование }
var X : byte;             { и вывод на экран }
begin
    n := 0;
    IF sym = '*' then begin X := 32; TextColor(LightRed) end
    else begin X := 20; TextColor(White) end;
    for k := 1 to 16 do begin
        W := 0;
        for i := 0 to 15 do IF Mem[$B800:N+i*2] = Byte(Sym) then
            Inc(W, 1 SHL (15-i));
        gotoXY(X,K); Write(W:6);
        Inc(N,160)
    end
end;

BEGIN
    TextMode(259); TextAttr := 31; ClrScr;
    for i := 1 to 16 do WriteIn('.....'); { растровая сетка }
    gotoXY(1,1);
    Repeat
        X := WhereX; Y := WhereY;
        M := ReadKey; TextColor(White);
        Case M of
{ ENTER } #13:          IF Y < 16 THEN gotoXY(1,Y+1);
{ забой }   #8:         begin gotoXY(X-1,Y);
                            Write('.'); gotoXY(X-1,Y) end;
{ пробел } #32:         IF X < 17 then Write('.');
           #33..#254: IF (X<17) then { проверка клавиши SHIFT }
                          IF (Mem[0:$0417] and 2) = 0 then
                          begin TextColor(White); Write('■') end
                          else begin
                              TextColor(LightRed); Write('*') end;
          #0:       begin m1 := ReadKey;
                        case m1 of
{ F1 }                      #59:  begin Print('*'); Print('■');
                                  gotoXY(X,Y) end;
{ HOME }                    #71:  gotoXY(1,Y);
{ клавиши }                 #80:  IF Y < 16 Then
                                  gotoXY(X,Y+1);
{ управления }              #72:  IF Y > 1 then
                                  gotoXY(X,Y-1);
{ курсором }                #77:  IF X < 17 then
                                  gotoXY(X+1,Y);
                            #75:  IF X > 1 then
                                  gotoXY(X-1,Y)
                        end
                    end
        end
    UNTIL m=#27;
    TextAttr := 7; TextMode(3); ClrScr
END.
```

■

Рекордов хотите? Их есть у меня

«Когда в клетке с надписью «слон» увидишь льва — не верь глазам своим», — так в свое время говаривал Козьма Прутков. Когда мы увидели слово «рекорд» рядом с программой распечатки текущих каталогов г-на Аброськина («Монитор» 5.93), то тоже не поверили своим глазам. Как программа, пространная более, чем доклад Генерального секретаря времен застоя, может претендовать на рекорд по компактности кода? Хотя мы не относим себя к членам KKK (не волнуйтесь, это не Ку-Клукс-Клан, а всего-навсего Клуб Кратких Кодов — см. «Софтмаркет» 20.93), неоптимальность данной программы была настолько вопиющей, что мы не могли спокойно пройти мимо. В лучшем случае данная «рекордная» программа может служить как отправная точка для оптимизации, выдаваемая в качестве учебного задания студентам, но не более. Каждому свое: кому-то постигать азы ассемблера, а кому-то устанавливать рекорды. При этом рекорды должны быть действительно рекордами, а не изобретением велосипеда.

Вот, например, г-н Аброськин в своей программе так получает адрес структуры, содержащей имена текущих каталогов:

```
mov di,es:[bx+16h]
mov ax,es:[bx+18h]
push ax
pop es
```

Хотя все это безо всякого ущерба можно заменить на

```
les di,es:[bx+16h]
```

Какая тайная цель преследовалась игнорированием этой команды, нам до сих пор не ясно, поскольку сегментный адрес ни в регистре AX, ни в стеке в дальнейшем не используется.

Или, скажем, другой пример необъяснимых решений. Как известно, в школе учат, что материя (дерево, пластмасса, hard) первична, а душа (soft) вторична. Но у г-на Аброськина отдается явное предпочтение soft. Иначе чем же объяснить трогательную заботу о версиях DOS младше четвёртой и в то же время полное игнорирование мониторов типа CGA или Hercules, что следует из использования 13h функции 10h прерывания.

Просто переписав выдачу информации на терминал через функции DOS и убрав явные «архитектурные излишества», мы сразу же сократили программу байтов этак на 36. Как известно, ИКАО (Международная аэрокосмическая организация) регистрирует новый рекорд, если он превосходит старый минимум на 10%. Вы сами можете подсчитать, что первая попытка превысила «рекорд» г-на Аброськина уже на 40%!

Движимые честолюбивыми мечтами, мы решили не останавливаться на достигнутом. Дальше, как говориться, работа пошла по секциям. В результате Д. Правиковым было предложено использовать для вывода символов конструкцию

```
lodsb
int 29h
```

А Д. Шумов разработал оптимальный алгоритм перехода к текущему каталогу очередного диска после вывода предыдущего каталога.

Дружное сокращение программы, больше напоминавшее совместное закрытие чемодана, в который положили все, что может пригодиться, привело к уменьшению ее размера до 46 байтов.

Все было бы хорошо, но смущало одно обстоятельство. Для вычисления в теле цикла смещения между каталогами в таблице использовалась следующая последовательность команд:

```
mov ah,30h
int 21h
mov cl,58h
cmp al,4
jae m1
mov cl,51h
m1: ...
```

Дело в том, что, помимо номера версии DOS в регистре AX, данная функция может в некоторых системах возвращать в регистрах BX и CX некоторое число. В принципе, этим можно было бы пренебречь, но сильнее боязни несовместимости было жгучее желание выкроить 1 байт и тем самым превысить «рекорд» г-на Аброськина ровно в два раза. И такое решение было найдено!

Д. Правиковым для получения нужной константы в CX в зависимости от версии DOS был предложен алгоритм, который кодируется следующим образом:

```
db 0D4h,04h
mov al,51h
db 0D5h,07h
xchg cx,ax
```

Это и позволило выкроить искомый байт.

Анализ того, как работает этот участок кода да и программа в целом, будет для читателей своеобразной проверкой на знание DOS и системы команд 8086. Авторы не претендуют на абсолютную оптимальность приведенной далее программы, но, как нам кажется, 45-байтовый вариант является близким к пределу. С тем большим удовольствием мы узнаем о дальнейшем ее совершенствовании.

Желаем успеха. Изучайте, сокращайте, используйте!

Д.И. Правиков, Д.Ю. Шумов, г. Москва ■

Листинг. Программа распечатки текущих каталогов.

```
0000                    code    segment
                        assume  cs:code,ds:code,es:code
                           org  100h
0100                    start:
0100  B4 52                mov  ah,52h
0102  CD 21                int  21h     ; "Get List of Lists"
0104  26: 8A 4F 20         mov  cl,es:[bx+20h]
0104                                    ; CX - количество
0104                                    ; дисковых устройств
0108  26: C5 77 16         lds  si,dword ptr es:[bx+16h]
010C                    lp1:
010C  51                   push cx
010D  B4 30                mov  ah,30h  ; "Get DOS version"
010F  CD 21                int  21h
0111  D4 04                db   0D4h,04h
0113  B0 51                mov  al,51h
0115  D5 07                db   0D5h,07h
0117  91                  xchg cx,ax
0118                    lp2:
0118  AC                   lodsb
0119  CD 29                int  29h     ; "DOS - Fast putchar"
011B  0A C0                or   al,al
011D  E0 F9                loopnz lp2
011F  B0 0A                mov  al,0Ah
0121  CD 29                int  29h
0123  B0 0D                mov  al,0Dh
0125  CD 29                int  29h
0127  03 F1                add  si,cx
0129  59                  pop  cx
012A  E2 E0                loop lp1
012C  C3                   ret
012D                    code    ends
                        end     start
```

■

Письмо сердитого читателя

Здравствуйте, уважаемая редакция!

Я недавно (с начала 1993 года) читаю ваш журнал. В целом он неплохой и представляет для меня большой интерес.

Это письмо написать меня побудили публикации А. Аброськина в рубрике «DOS под микроскопом» (в которых он учит начинающих программистов писать более-менее сносные TSR). То, о чем пойдет речь, относится, в частности, к «Монитору» № 1, 2, 3 за 1993 год. Дело в том, что при более подробном рассмотрении приведенных листингов я обнаружил общую грубейшую ошибку (логическую!), которая не только сводит на нет все усилия по набивке и отладке программы (это уж ладно), но и пахнет «смертельным исходом, платой за который будет в лучшем случае нажатие клавиши «RESET», ну а в худшем... испорченное настроение окружающих и кропотливая работа по восстановлению утраченных данных».

А именно. Рассмотрим ситуацию, когда резидент уже установлен в памяти и я хочу его выгрузить. При запуске с соответствующим ключом программа читается в память и ей передается управление. (Для конкретности изложения пусть это будет **Game$.com**.) Переменные (назову их так) **Old21s** и **Old21o** не инициализированы и по изначальному определению равны нулю. Программа же, распознав ключ выгрузки, ставит в таблицу векторов прерывания 21h эти переменные, которые равны нулю!!! Таким образом, получается, что адрес вектора 21h есть 0000:0000. При первом же вызове 21-го прерывания управление будет передано по этому адресу, то есть на таблицу векторов прерываний! Четырехбайтовые адреса векторов будут трактоваться как инструкции микропроцессору, и он их будет послушно исполнять!! Хорошо, если все закончится зависанием системы (что наиболее вероятно), но ведь можно потерять все данные на жестком диске и сделать компьютер незагружаемым даже с дискеты!! Да, да, не удивляйтесь. Хотя по теории компьютер пытается загрузиться с дисков по порядку (от «A» до «Z»), практика показывает, что у некоторых правил есть большой процент исключений, и я не раз сталкивался с тем, что при испорченной операционной системе на винчестере (all data, естественно, lost) машина не загружается с **безупречной** системной дискеты, хоть первоначально и обращается к ней. (Имею информацию по поводу создания и удаления таких казусов).

Итак, программа подставила адрес прерывания 21h 0000:0000. И сразу же после этого она дважды (!) вызывает это 21-е прерывание (первый раз при вызове функции 49h — «освободить блок памяти», а второй — при вызове функции 4Ch — «завершить программу»).

Это ли не вредительство?!! Я считаю, что это подлость по отношению к начинающим и неискушенным программистам, которые на ваших примерах пытаются постичь азы ассемблера.

Конечно, может, я неправ, но тогда пусть мне г-н А. Аброськин скажет, каким образом в данные вновь запущенной копии программы попадет адрес старого (оригинального) обработчика 21-го прерывания? Я не люблю вирусы, поэтому и не пишу их, но приведенные листинги чем-то смахивают на них. Может быть, потому что пользователь (начинающий программист), сам того не зная, «роет себе могилу», что, собственно, и происходит при запуске зараженных программ.

Предлагаю вариант. Все резидентные программы, предусматривающие выгрузку из памяти, должны перехватывать, кроме других, еще и функцию 21h и проверять на функцию «Exec» (4Bh — прим. ред.). И если была запущена копия резидента, то проверить командную строку на ключ выгрузки. Если есть, то «терминировать» запуск копии и выгрузить себя из памяти.

Другой вариант. Зная адрес нашего резидента (из таблицы векторов прерываний) и относительный адрес (относительно точки входа в процедуру обработки прерывания) этих двух «переменных», читаем их содержимое — сегмент и смещение оригинального обработчика, которые мы потом и подставляем (при выгрузке) в таблицу векторов.

Ни один из этих вариантов не реализован, и, как и следовало ожидать, выгрузить резидент из памяти удалось только после «холодного» рестарта системы. (Мы рискнули потерять данные и запустили с ключом выгрузки).

Мое предположение, возможно, ошибочно, но тем не менее я прошу вас высказать свое мнение и узнать мнение Аброськина по этому поводу, и выслать мне ответ, если, конечно, это вас не затруднит. Если хотите, можете опубликовать письмо (целиком или фрагменты), но, как бы то ни было, я с нетерпением жду вашего ответа и особенно ответа Аброськина или публикации на эту тему в следующих номерах «Монитора».

С уважением, А. Сулимов, г. Омск ■

Ответ редакции

Надо признать, письмо г-на Сулимова нас озадачило. Мы стараемся проверять все программы, которые присылают нам наши авторы (в частности, для того чтобы избежать опечаток, мы требуем от авторов присылать статьи на машинном носите-

ле). Статьи г-на Аброськина, о которых шла речь, набивала наша машинистка, а поэтому мы и проверяли их очень тщательно, отлаживая набитый текст самостоятельно. *Конечно же, мы проверяли все описанные режимы. Мы согласны, что от синтаксических ошибок не застрахован никто, но ЛОГИЧЕСКИЕ? Поэтому мы переслали письмо г-ну Аброськину и попросили его ответить автору...*

В. Водолазкий, Г. Карчикян ■

Ответ сердитому читателю

Ну что же, давайте поговорим о взаимодействии TSR-программы с операционной системой и о программировании на ассемблере, в частности. Прежде чем предметно перейти к существу вопроса, позволю себе привести фрагмент кода, на который ссылается читатель. Итак, в транзитной части программы присутствуют следующие строки, отвечающие за выгрузку программы из памяти и восстановление «захваченного» вектора прерывания:

```
    mov   ax,3521h   ; Получаем адрес вектора int 21h
    int   21h        ; функцией DOS "Get Interrupt Vector"
    mov   ax,es:[bx-2]    ; Проверяем ключевое слово
    jnz   load            ; обработчика
    jmp   already         ; Если совпадает, то...
load:
    ....
    ....
already:                  ; Проверяем командную строку на
    ....                  ; ключ выгрузки, если есть, то
    ....
unload:                                ; Вычисляем адрес
    mov   ax, word PTR es:Old21o       ; "старого" вектора
    mov   dx, word PTR es:Old21s       ; используя значение ES !!!
    mov   cx, es  ; Сохраняем сегментный адрес обработчика
    cli           ; int 21h и запрещаем прерывания
    xor   bx, bx
    mov   es, bx          ; Устанавливаем ES:[BX] на ячейку
    mov   bx, 21h*4       ; памяти, используемую для хранения
                          ; вектора 21h
    mov   es:[bx], ax     ; Восстанавливаем оригинальный
    mov   es:[bx+2], dx   ; адрес вектора 21h
    sti                   ; Разрешаем прерывания
    mov   ah, 49h         ; Перед вызовом функции "Free
    mov   es, cx          ; allocated Memory" устанавливаем ES
    int   21h             ; на сегментный адрес обработчика int 21h
    mov   ax, 4C00h
    int   21h             ; Выход в DOS
```

Давайте последовательно рассмотрим этот фрагмент. Сначала вызывается функция DOS «Get Interrupt Vector», которая возвращает в регистры ES и BX адрес запрашиваемого вектора. Затем в регистр AX заносится значение СЛОВА, находящегося перед точкой входа в обработчик прерывания и после этого осуществляется сравнение этого СЛОВА со СЛОВОМ, находящимся в сегменте кода вновь запущенной программы. Если обнаружено совпадение,то осуществляется переход на метку ALREADY, где производится синтаксический разбор командной строки, и в случае обнаружения ключа выгрузки выполняется переход на метку UNLOAD. Ну а теперь мы подошли к самому главному.

Отвечаю на Ваш вопрос: «Каким образом в данные вновь запущенной копии попадет адрес старого (оригинального) обработчика 21-го прерывания?»

Никогда, и ни под каким соусом! (если Вы конечно, имеете в виду переменные old21s и old21o). А вот ежели Вас интересует, каким же образом программа сможет узнать адрес «старого» вектора и куда оный будет помещен, то предлагаю Вам внимательно посмотреть на строки кода, начиная с метки UNLOAD, в которых черным по белому написано, что адрес «старого» вектора заносится в регистры AX и DX, из которых он (то бишь адрес) и будет «подставлен» в таблицу векторов. Сразу же обращаю Ваше внимание на то, что инструкция MOV AX, word PTR ES:old21o «предписывает» ассемблеру переслать в регистр AX содержимое СЛОВА, адресуемого указателем типа СЕГМЕНТ:СМЕЩЕНИЕ. А как Вы помните, ES к этому времени уже содержал сегментный адрес вектора 21h, а вот old21o — всего лишь метка, вместо которой ассемблер при генерации машинного кода подставит значение, равное ее относительному смещению. Кроме того, задумайтесь как-нибудь на досуге, для чего используется так называемый атрибутный оператор указателя (PTR) и как он влияет на состояние бита W в поле кода операции. Приводимые далее по тексту инструкции не требуют больше разъяснений, так как любой школьник, освоивший функции DOS и несколько инструкций ассемблера, разъяснит Вам происходящие далее процессы.

Думаю, что Ваша ошибка «зарыта» в «набитом» тексте программы, и, по всей видимости, Вы перепутали сегментные регистры. Хотя, как следует из Вашего письма, Вы исследовали листинги моих программ аж в целых трех номерах «Монитора» и нашли там общую грубейшую логическую ошибку, а следовательно, мне с сожалением приходится констатировать, что Вы недостаточно изучили ассемблер, формат машинных команд и не представляете себе разницу в способах определения операнда, то есть режимов адресации микропроцессора.

Небольшой комментарий по поводу предлагаемых Вами вариантов выгрузки резидента из памяти. Позвольте теперь мне задать Вам пару риторических вопросов. Интересно узнать, каким образом, перехватывая функцию «Exec», Вы будете проверять, тот ли резидент должен быть выгружен из памяти? По имени программы? Или по ее контрольной сумме? И не надо ли в таком случае иметь еще отдельную резидентную процедуру, «машущую» перед копией резидента флагом по имени Carry и имеющую функции по освобождению занимаемой памяти и восстановлению захваченных векторов прерываний? И как это безобразие будет влиять на размер резидентной программы? Ответив на эти вопросы, Вы можете смело учить начинающих программистов «писать более-менее сносные TSR». Ну, а что касается второго варианта, то мне жаль, что Вы не разглядели его реализацию в исследуемых Вами листингах моих TSR.

Ну, и последнее, по поводу вирусов. Любая команда «обычной программы» может являться и работоспособной частью кода вируса, дело лишь в том, в каком контексте ее рассматривать. Многие резиденты, да и вирусы тоже, используют нетривиальные способы взаимодействия с операционной средой, а потому могут иметь и некоторые общие черты, так тонко «увиденные» Вами в листингах TSR. Однако советую Вам не заниматься этой тематикой, так как для этого неплохо было бы изучить ассемблер, систему и знать еще технику разработки подобного матобеспечения, иначе Ваш компьютер при «all data, естественно, lost», будет незагружаемым даже с гибкого диска, о чем так красноречиво свидетельствует накопленная Вами информация по поводу создания и удаления таких казусов. Что, впрочем, не мешает Вам, не откладывая на завтра, отправиться в магазин, приобрести хорошую книжку по ассемблеру и погрузиться с большой пользой для себя в этот волшебный мир сегментов и смещений.

А. Аброськин, г. Нижний Новгород ■

От редакции

В статье «SIG Paradox» Л. Ливеровского, опубликованной в «Мониторе» 5.93, допущена ошибка: вместо рис. 2 повторно напечатан рис. 1.
Редакция приносит автору и читателям свои извинения.

Наш книжный ларёк

Издательство «Диалектика» (Киев)

1. Гради Буч. Объектно-ориентированное проектирование с примерами применения	2800
2. Object Windows для C++ (в 2 томах)	1500
3. Turbo Vision для языка Паскаль (в 2 томах)	1500
4. А. Федоров, Д. Рогаткин. Borland Pascal в среде Windows	4500
5. А. Федоров. Borland: практическое использование Turbo Vision 2.0	2000
6. Resource Workshop. Проектирование ресурсов Windows-приложений	2200

Издательство «Кварта» (Москва)

Справочник программиста и пользователя/ Под ред. А.Г. Шевчика, Т.В. Демьянкова	750

Цены приведены в рублях с учетом стоимости пересылки. Наш расчётный счёт 467996 в Независимом Банке России, к/с 1161385 в РКЦ-2 ГУ ЦБ РФ по г. Москве, МФО 201779.
Ваши заявки и квитанцию почтового перевода отправляйте по адресу: 117419 Москва, а/я 765.

Отечественные CASE–средства разработки проектов информационных систем

А. Морозова

1. Основы CASE-подхода

Подход CASE (Computer Aided System Engineering), созданный для компьютерной поддержки жизненного цикла (ЖЦ) создания программных систем, к концу 80-х годов оформился в самостоятельное и перспективное направление программной инженерии. В рамках данного направления важное место занимают работы по автоматизации разработки проектов информационных систем (ИС), которые все шире используются в различных сферах человеческой деятельности [1]. «Время-качество-простота» составляют триаду тех выгод, которые получает проектная бригада при использовании CASE-технологии.

Реализация CASE-технологии проектирования ИС предусматривает выбор методологии проектирования и создание инструментальных средств для оснастки АРМ проектировщиков, работающих над общим проектом. В методологии определяются последовательность шагов разработки, состав проектных спецификаций на каждом шаге, методы их формализации, анализа и преобразования, а также распределение функций (ролей) между проектировщиками.

В зависимости от вида проектных описаний, специфицируемых в методологиях, они делятся на структурные, функциональные и смешанные, в том числе объектно-ориентированные. Структурные методологии определяют структуру информационного фонда ИС вне зависимости от процессов его обработки. В функциональных методологиях первичными являются процессы (функции или активности), которые последовательно детализируются до уровня программных модулей. Третья группа методологий предлагает параллельное взаимосвязанное проектирование структур данных и процессов их обработки.

Методологии классифицируют и по их принадлежности этапам ЖЦ создания ИС на методологии верхнего, среднего и низкого уровней [2]. Традиционно этапами ЖЦ стадии проектирования ИС являются стратегическое планирование, анализ требований, концептуальное, логическое и физическое проектирование. Следующая затем стадия реализации ИС включает этапы кодирования информационных задач, установку системы, ее тестирование и сопровождение. Методологии верхнего уровня предназначены для этапа стратегического планирования. Методологии для этапов анализа и проектирования относятся к среднему уровню. Низкий уровень объединяет методологии для стадии реализации.

К настоящему времени сложился стандарт де-факто на формальные средства анализа и проектирования, введенные различными методологи-

ями и широко используемые в программном инструментарии проектировщиков. К ним относятся разнообразные диаграммы (потоков данных и управления, переходов состояний, зависимостей и декомпозиции, Варнье-Орра и SADT-диаграммы, произвольные иллюстративные диаграммы), структурные карты, таблицы и деревья решений, макеты экранных форматов, псевдоязыки для описания на семантическом уровне логики процессов, а также матрицы перекрестных ссылок между разнотипными спецификациями в составе проекта.

Как правило, инструментальные CASE-средства реализуют методы одной или нескольких методологий, но могут быть построены и на специализированных подходах, не формализованных в виде самостоятельной методологии.

ТЕХНОЛОГИИ

У нас практика разработки ИС во многом использует «ручные» методы проектирования и традиционный бумажный способ накопления проектных спецификаций.

Современные CASE-средства реализуют следующие группы функций.

1. Ведение на магнитном носителе всех видов проектных описаний для ЖЦ разработки в режиме их «ручного» определения (через графические диаграммы и/или экранные формы).
2. Верификация проектных спецификаций.
3. Документирование.
4. Автоматическая генерация некоторых видов проектных описаний.
5. Спецификация интерфейса с будущими пользователями ИС в виде создания экранных форматов планируемого представления информации.
6. Возможность параллельной работы проектировщиков на разных АРМ над общим проектом.
7. Разграничение доступа проектировщиков.
8. Простой и удобный интерфейс.

Первоначально опыт поддержки этих функций был получен на больших ЭВМ. Только последние достижения персональной компьютерной техники создали среду для эффективной реализации удобных в эксплуатации CASE-средств. В эту среду вошли персональные СУБД с их языками 4-го поколения (4GL), графический и текстовый редакторы, режимы сетевой работы и новые оконные интерфейсы.

Несмотря на разнообразие существующих в мире CASE-средств, все их роднит использование компьютерного хранилища (репозитория, базы данных проекта, словаря проектирования) для накопления проектных спецификаций на машинном носителе. Современный репозиторий должен обеспечивать хранение формальных проектных характеристик, неформализуемых описаний в виде текстов, а также различных диаграмм и матриц. Многочисленные и неизбежные корректировки проекта в итеративном по своей природе процессе проектирования перестают быть проблемой, когда бумажный носитель проектных спецификаций заменен на электронный. Для управления базой данных (БД) проекта используют, как правило, одну из современных СУБД, функционирующих на ПЭВМ.

«Ручное» определение проектных спецификаций, хранимых на машинном носителе, а также их автоматическая верификация и документирование составляют тот минимальный уровень автоматизации, с которого начинается практическая польза от применения инструментальных средств. «Большая» CASE-автоматизация включает генерацию различных видов производных спецификаций, прежде всего логической структуры БД, ее объемов, привязку к файлам внешней памяти и так далее. К производным спецификациям относятся и различные виды матриц, анализ которых помогает проектировщику в принятии ряда проектных решений.

Перечисленными функциональными возможностями обладают CASE-средства первого поколения, созданные в течение 80-х годов. В последние годы появился инструментарий нового, второго поколения. В его состав вошли средства кодогенерации целевой ИС по накопленным проектным спецификациям и средства реверсной инженерии (реинжиниринга, то есть восстановления из программного кода ИС в хранилище CASE-средств отдельных видов проектных описаний).

Средства кодогенерации большинства CASE-продуктов ориентированы на единственную СУБД или язык программирования, что заставляет проектировщиков, еще только приступая к проекту, выбрать среду реализации будущей ИС. В таком инструментарии целевая СУБД входит в его состав и используется для управления компьютерным хранилищем проекта, а в спаринге «CASE — СУБД» создается единая среда проектирования и реализации ИС.

2. Состояние отечественной CASE-технологии

В нашей стране практика разработки проектов ИС, к сожалению, во многом использует «ручные» методы проектирования и традиционный бумажный способ накопления проектных спецификаций. Это можно объяснить малым числом отечественных CASE-продуктов и небольшим количеством публикаций, популяризирующих CASE-подход.

Несмотря на наличие на Западе нескольких десятков промышленных CASE-средств, на нашем рынке из них доступны единицы, прежде всего *CASE*Oracle*, *Prokit*Workbench*, *MetaDesign* и *Design IDEF* [3]. Высокая валютная стоимость этих продуктов, отсутствие русифицированных версий, поддержка зарубежных стандартов проектных документов, а также ориентация на более мощные ПЭВМ, чем получившие распространение у российских пользователей, — вот причины, которые способствовали активизации собственных разработок в данном направлении.

Центральный Российский Дом Знаний с 1992 года начал проведение ежегодного семинара «CASE-технология», по докладам которого можно судить о доступных у нас в стране зарубежных CASE-средствах и новых отечественных разработках. Зарегистрированная в марте 1993 года российская ассоциация «Программная инженерия» будет способствовать научным контактам специалистов, работающих в данном направлении. В этом же году выходит в свет первая отечественная монография [5], в которой предложена методология проектирования ИС в рамках объ-

ектно-ориентированного подхода для всего ЖЦ разработки. В ней приведен обзор современных подходов к концептуальному моделированию предметных областей, для этапов анализа, макетирования (прототипирования) и проектирования предложены формальные методы, ориентированные на реализацию в инструментальных средствах. В монографии рассмотрены также вопросы реализации ИС в среде ряда известных персональных СУБД (*dBASE*, *FoxPro*, *Clipper*, *Paradox*, *Clarion*) и отечественной СУБД *ДИСОД*, предназначенной для функционирования на больших ЭВМ.

Далее в статье дается краткая характеристика десяти инструментальных средств, сведения о которых доступны из публикаций или рекламных буклетов программных выставок последних лет. В состав анализируемого списка вошли следующие CASE-средства:

АРИУС и *АРИУС-ПК* (Институт проблем управления, Москва);
ИРИС (ВНИИ системных исследований, Москва);
Дизайнер баз данных (Университет, Санкт-Петербург);
СИНТЕЗ (Академия им. Жуковского, Москва);
КОМОД (Институт кибернетики, Киев);
ИКС (НИИ «Восход», Москва);
СЛОВАРЬ ПРОЕКТА (Научно-производственная фирма «ТЕХНОСОФТ ТЕСТ», Белгород);
CASE.Аналитик (Научно-техническое предприятие «Эйтекс», Москва);
*CASE*ДИАГРАММЕР* (РосНИИ ИТ и АП, Москва);
МАГИС (ИСА РАН, Москва).

Системы *АРИУС*, *ИРИС* и комплекс программ концептуального проектирования баз данных *ПРОРОК* в составе инструментальных средств *СИНТЕЗ* были созданы в середине 80-х годов для эксплуатации на больших ЭВМ. Остальные CASE-средства из приведенного списка функционируют на IBM совместимых ПЭВМ. За исключением *CASE.Аналитик* и *CASE*ДИАГРАММЕР*, требующих операционной системы WINDOWS, основная часть инструментальных средств работает в среде MS-DOS.

АРИУС и *МАГИС* являются инструментальной надстройкой над собственной «материнской» СУБД, которая используется как для управления проектным хранилищем, так и для реализации проектируемой ИС. При создании ИС этими средствами ее генерация происходит автоматически по накопленным спецификациям, основу которых составляют структуры входных и выходных документов. Оба инструментальных средства основаны на подходе, получившем название «снизу вверх», при котором логическая структура БД определяется путем интеграции локальных подсхем, специфицируемых в виде структур документов.

Роль СУБД для *АРИУС* играют программы в составе компонента, названного оперативным комплексом. CASE-средства *МАГИС* базируются на отечественной СУБД *НИКА* для ПЭВМ. Другие отечественные CASE-продукты для управления БД проекта используют автономную СУБД, главным образом, *Paradox*, *Clipper* и *dBASE*.

Зарубежные методологии структурного анализа и концептуального моделирования положены в основу *CASE.Аналитик*, *CASE*ДИАГРАММЕР* и *СЛОВАРЬ ПРОЕКТА*. Инструментальные средства *ИКС*, *Дизайнер баз данных*, *СИНТЕЗ* и *КОМОД* используют авторские методы формализации, анализа и преобразования проектных описаний. Авторы CASE-средств *Дизайнер баз данных* и *ИКС* описали предлагаемые ими методы проектирования в отдельно выпущенных книгах [4], [5], другие авторы — в статьях и докладах [6] — [14].

3. Краткая характеристика CASE-средств

АРИУС и АРИУС-ПК. Система *АРИУС* (более поздняя версия для ПЭВМ называется *АРИУС-ПК*), выпущенная в 1980 году, состоит из инструментального и оперативного комплексов [6]. Она поддерживает методологию *АРИУС* (автоматизация разработки информационно-управляющих систем), которую по уровневой классификации следует отнести к методологии среднего и низкого уровней.

Описание исходных требований ведется на языке табличного вида *АРИУС-80*, позволяющем специфицировать до 400 документов, включая внешний эскиз документа, состав его реквизитов и переменных. В описание переменной входит формат ее представления, формула вывода значений и их ожидаемое число, а также ключ и ограничения целостности. Формула может быть задана посредством элементарных функций языка *АРИУС* или как таблица решений.

Для оперативного комплекса логическая структура БД и программы, выполняющие ввод, корректировку и преобразование входных документов в выходные, генерируются автоматически. После генерации ИС на стадии ее эксплуатации возможно итеративное внесение изменений в проект с последующей автоматической корректировкой программ ИС.

Сфера применения *АРИУС* ограничена классом задач экономико-организационного типа.

ИРИС. Система *ИРИС* (интерактивная разработка информационных систем) была разработана в середине 80-х годов для эксплуатации на машинах серии ЕС ЭВМ и СМ ЭВМ. (Ее не следует путать с одноименной отечественной СУБД для ПЭВМ, созданной для управления БД документов с произвольной иерархической структурой.) Для управления БД проекта в данной системе использована русифицированная версия СУБД *Oracle*, получившая название *АККОРД*.

ИРИС поддерживает в ЖЦ три этапа: концептуальное проектирование предметной области, логическое проектирование БД и физическое проектирование БД. Инструментальные средства этих этапов оформлены в виде подсистем *КОМОД* (КОнцептуальное МОДелирование), *СИЛОС* (СИнтез ЛОгической Структуры) и *ОФИС* (Оптимизация ФИзической Схемы). Главное назначение *ИРИС* — автоматизация проектирования логической и физической схем баз данных для реляционных СУБД.

Формальные методы *ИРИС* относятся к методологии структурного типа, основанной на подходе «снизу вверх». Концептуальная схема предметной области создается путем интеграции локальных представлений пользователей. Локальные представления и концептуальная схема специфицируются в виде множества функциональных и многозначных зависимостей, названных семантическими отношениями. Данное множество зависимостей используется для

синтеза логической схемы БД, который можно выполнить по алгоритму Бернштейна или модифицированному алгоритму Делобеля-Кейси.

На первом этапе вводится вся исходная информация для последующего проектирования. В ее состав входят пять видов проектных описаний: *ПРОЕКТ*, *ЛОКАЛЬНОЕ ПРЕДСТАВЛЕНИЕ*, *АТРИБУТ*, *СЕМАНТИЧЕСКОЕ ОТНОШЕНИЕ* и *ЗАПРОС*. Для каждого описания, помимо специфических характеристик, определяются имя (английское и русское), однострочный комментарий и текст произвольной длины. На этом же этапе происходит интеграция локальных представлений, в результате чего создается файл с множеством зависимостей между атрибутами.

Инструментальные средства второго этапа состоят из 12 функций, с помощью которых можно исследовать структуру семантических отношений и синтезировать логическую схему БД. На этапе физического проектирования решаются три оптимизационные задачи: размещение атрибутов по типам записей (в случае необходимости дробления больших отношений), определение набора кластеров записей и выбор состава вторичных индексов при задаваемом ограничении на внешнюю память. Оптимизация проводится по критерию минимального времени обработки наиболее важных для ИС запросов, названных регламентированными, описания которых вводятся на первом этапе проектирования. В основе всех видов оптимизации лежит использование матрицы толерантности, каждый элемент которой содержит меру толерантности, вычисляемую как функцию длин атрибутов и частоты их совместного использования в регламентированных запросах.

Дизайнер баз данных. Данные CASE-средства созданы для этапов анализа требований и концептуального проектирования БД [8]. Реализуемый методологический подход изложен в [4]. В БД проекта можно помещать 11 видов проектных описаний: *ФРАГМЕНТ*, *ПРОЦЕСС*, *ПОЛЬЗОВАТЕЛЬ*, *ДОКУМЕНТ*, *РЕКВИЗИТ*, *ДОМЕН*, *ТИП*, *ОБЪЕКТ*, *АТРИБУТ*, *ЗАПРОС* и *АССОЦИАЦИЯ*.

Для концептуального моделирования структуры предметной области предложена модель данных, названная «объекты-связи». В ней предусмотрены агрегация атрибутов и обобщение объектов, а также связи, определяемые через атрибуты-ссылки и составные ключи объектов. Объекты образуются путем сборки атрибутов в результате анализа общих доменов и состава документов предметной области. Функциональные аспекты анализа требований отражены в определении процессов, запросов и ассоциаций структур данных, входящих в состав структурной подсхемы запросов.

Предложенная технология проектирования включает последовательность следующих шагов:
- установление связи между реквизитами и доменами;
- агрегация доменов и определение объектов;
- идентификация первичных ключей объектов;
- кодификация пользователей;
- обобщение объектов и нормализация;
- трансформация запросов;
- генерация логической схемы на языке описания данных (ЯОД) выбранной СУБД;
- верификация проекта и его документирование.

СИНТЕЗ. Это семейство инструментальных средств *ПРОРОК*, *СИНТЕЗ-Т* и *СИНТЕЗ+* [9]. Созданные для среды ОС ЕС ЭВМ средства *ПРОРОК* объединяют программы синтеза нормализованных отношений по множеству исходных функциональных зависимостей. Два следующих продукта в составе семейства *СИНТЕЗ* работают в среде MS-DOS на персональных ЭВМ. Их функциональный состав отражает эволюцию развития программ логического проектирования БД, реализованных в составе средства *ПРОРОК*, с целью поддержки многоэтапной структурно-функциональной методологии проектирования ИС.

Средства проектирования *ПРОРОК* осуществляют ввод информации о структурных элементах, формализуя ее в виде множества функциональных зависимостей (ФЗ), и накапливают понятийный состав для создания словарей и классификаторов информационного и лингвистического обеспечения ИС.

Над множеством ФЗ реализовано выполнение следующих функций:
- определение связных подсистем для произвольной системы ФЗ;
- определение замыкания любой исходной системы ФЗ;
- выделение семейства минимальных покрытий для любой связной системы ФЗ;
- определение существенных детерминант одного из минимальных покрытий исходной системы ФЗ.

Функциональный состав средств *ПРОРОК* позволяет проектировщику исследовать структуру ФЗ и выбирать различные стратегии формирования концептуальной схемы БД. Существует возможность формирования семантически эквивалентных (в том числе и оптимальных) концептуальных схем БД и соответствующих им алгоритмов процедур контроля целостности и согласованности с количественной оценкой их сложности.

Технологический инструментальный комплекс поддержки разработки систем баз данных *СИНТЕЗ-Т* адресован проектировщикам малых и средних ИС. Для описания структуры предметной области в нем используются формализованные предложения естественного языка, преобразуемые для дальнейшего анализа в семантическую сеть фреймов и множество ФЗ, передаваемое программам синтеза концептуальной схемы БД. Для этапа логического проектирования реализованы функция отображения концептуальной схемы в реляционную структуру и функция генерации программного файла с описанием на ЯОД логической схемы для СУБД *Clipper*. В отличие от средств *ПРОРОК* имеется более развитая подсистема документирования проектных решений.

Последняя версия данного семейства *СИНТЕЗ+* предназначена для поддержки стадии проектирования средних и больших ИС и объединяет этапы от стратегического планирования до физического проектирования. Для описания структуры предметной области используется модель данных, названная авторами «объекты-свойства», основу которой составляют иерархии агрегации и обобщения. Реализована технология параллельной работы проектировщиков с последующей интеграцией частных пользовательских структур в общую структуру предметной области.

В состав *СИНТЕЗ+* входят подсистема администрирования проекта его руководителем, средства управления версиями проекта (*Архиватор*), экспорта-импорта, генерации отчетов и выдачи справочной помощи. Алгоритмические методы генерации проектных спецификаций, предложенные методологией *СИНТЕЗ*, реализованы в составе специализированных подсистем этапов концептуального, логического и физического проектирования. К ним относятся *Анализатор/Интегратор* (построение структуры предметной области), *Нормализатор/Оптимизатор* (синтез и многокритериальная оптимизация схем БД), *Синтезатор распределенных баз данных*, *Генератор схемы БД* на ЯОД выбранной СУБД.

Средства прототипирования задач ИС входят в состав подсистемы *«Конструктор пользовательской среды»*, позволяющей специфицировать внешние представления документов и сценарии диалога информационных задач.

КОМОД. Система *КОМОД 2.0* представляет собой инструментальные средства этапа концептуального проектирования [10]. Для описания структуры предметной области в ней используется концептуальная модель данных, компонентами которой являются классы и атрибуты. Классы можно объединять в многоуровневые иерархии обобщения. Связи между классами описываются через атрибуты с составным доменом значений.

При построении иерархии обобщения задаются предикаты подклассов, которые специфицируют условия принадлежности к экземплярам подкласса. В состав предиката могут входить константы, имена атрибутов класса и его суперклассов, а также арифметические и логические операции, операции сравнения и функции интегральных вычислений (среднее, максимум и так далее).

Для каждого класса и атрибута допустимо определение текста, хранимого в текстовом файле, и графического изображения, подготовленного пакетом *STORYBOARD PLUS*. Эти описания названы гипертекстовым справочником, по которому возможна навигация.

Для работы с концептуальными схемами реализованы функции ведения описаний их компонентов, навигации, попарной интеграции концептуальных схем, а также функция анализа последствий корректировки схемного определения атрибута.

Несмотря на узкий спектр охвата функций ЖЦ и специфичную концептуальную модель данных, для практики отечественной CASE-технологии *КОМОД* интересен проработкой алгоритма интеграции концептуальных схем с многоуровневыми иерархиями обобщения и связями через внешние ключи, а также возможностью специфицировать сложные условия вхождения подклассов в специализации классификационных деревьев, которые, как правило, не поддерживаются при реализации широко используемой модели данных «сущность-связь» (ER-модели).

ИКС. Инструментальные средства *ИКС* (инструмент конструирования систем), аналогично семейству САПР *СИНТЕЗ*, поддерживают этапы ЖЦ от стратегического планирования до физического проектирования [5].

Для моделирования предметной области на начальных этапах ЖЦ в системе *ИКС* применен объектно-ориентированный подход, специфицирующий ее поведение в виде событий (с пост- и предусловиями) и сообщений, передающих множество аргументов и инициирующих методы обработки объектов. В рамках этого подхода функциональные возможности *ИКС* пересекаются с возможностями CASE-средств, создаваемых для проектирования систем реального времени.

В состав подсистемы макетирования информационных задач входит *Редактор эскизов* (*Рисовальщик*), фиксирующий внешние представления документов пользователей. Имеются средства прокрутки сценариев диалога и анализа состояний предметной области в зависимости от сочетания начальных условий.

Для спецификации концептуальных схем БД предложена классификационно-динамическая модель данных (КД-модель). КД-модель структурирует данные в виде объектов, состоящих из иерархически подчиненных агрегатов, причем объекты и/или их агрегаты можно включать как классы в состав классификационных деревьев. Специальный вид метаданных «Классификационная решетка» введен для спецификации пересечения классов различных специализаций одного и того же суперкласса. Для концептуальных схем определена функция их попарной интеграции и последовательного согласования схемы со множеством взаимодействующих с ней процессов обработки данных.

Инструментарий *ИКС* для этапа логического проектирования имеет средства настройки на модели данных СУБД как с плоской (реляционной), так и иерархической структурой. Это позволяет преодолеть ограничения большинства CASE-средств, способных генерировать схемы только для реляционных СУБД. В рамках *ИКС* можно создавать логические схемы на основе разных моделей данных, что учитывает практику реализации больших ИС с разными типами СУБД.

ИКС накапливает в своем хранилище, названном словарем проектирования, более 20 видов проектных описаний и реализует построение различных видов матриц для анализа перекрестных ссылок между разнотипными метаданными.

СЛОВАРЬ ПРОЕКТА. CASE-средства, названные *СЛОВАРЬ ПРОЕКТА*, поддерживают этапы концептуального и логического проектирования БД в рамках чисто структурного подхода [11]. Для описания концептуальной схемы предметной области применена модель данных «сущность-связь», формализуемая в соглашениях созданной для *CASE*Oracle* методологии *CASE*Method*. Имеются встроенные средства рисования ER-диаграмм небольшого объема.

На этапе логического проектирования автоматически генерируется схема БД для реляционных СУБД в виде множества нормализованных реляционных таблиц. Полученную схему можно корректировать и дополнять характеристиками с учетом синтаксиса ЯОД целевой СУБД, после чего состав схемы выводится в виде программного файла на языке PAL СУБД *Paradox*. Реализованы функции контроля качества и несложные отчеты о составе проектных описаний, а также обратное преобразование ЯОД описания в словарные спецификации реляционных таблиц и концептуальной схемы (реинжиниринг описания БД).

CASE.Аналитик и CASE*ДИАГРАММЕР. Эти инструментальные средства созданы для структурного анализа проектируемых систем, используют среду WINDOWS и имеют развитые *Редакторы графических диаграмм*. В них использована методология, фиксирующая функциональное описание ИС в виде диаграмм потоков данных в нотации Gane-Sarson [12], [13].

Помимо диаграмм потоков данных, которые делятся на информационные и управляющие, в *CASE.Аналитик* [12] возможно построение структурограмм, описывающих состав накопителей данных и структур данных, передаваемых информационными потоками. Для элементарных данных определены четыре типа значений: данные в диапазоне или из дискретного списка, а также аналоговые и дискретные сигналы.

В *CASE*ДИАГРАММЕР* [13], помимо этапа анализа требований, поддержано проектирование концептуальной и логической схем БД. Для ER-моделирования, аналогично инструментальным средствам *СЛОВАРЬ ПРОЕКТА*, использованы формальные подходы методологии *CASE*Method* фирмы Oracle Corp. Возможно построение диаграмм «сущность-связь» средствами графического редактора, удовлетворяющего современным требованиям техники построения диаграмм. Средства генерации реляционных схем создают программные файлы с описанием таблиц на языке SQL или языке PAL для СУБД *Paradox*.

Оба инструментальных средства тиражируют на хорошем современном уровне технологические решения, положенные в основу ряда зарубежных CASE-продуктов. Однако несомненным достоинством *CASE.Аналитик* является развитая подсистема документирования (*Документатор*), созданная для поддержки отечественных нормативных требований к проектной документации. Средства *Документатора* позволяют создавать целый ряд макетов документов в соответствии с требованиями ГОСТ 19.XXX и ГОСТ 34.XXX. При выводе диаграмм на печать можно автоматически генерировать рамку и надписи по ЕСКД.

МАГИС. Система *МАГИС* (МАкетный Генератор Информационных Систем) создана для автоматической генерации ИС, функционирующих в среде СУБД *НИКА*, по макетам (формам) входных и выходных документов [14]. Отличительной особенностью *МАГИС* является отсутствие каких-либо ограничений на структуру документов (число уровней иерархии, данных на каждом уровне, индексируемых полей).

Для заданного множества форм документов *МАГИС* создает ИС с диалоговым вводом и корректировкой документов и запросной подсистемой. Ввод данных возможен как в БД, так и в файлы MS-DOS в формате, названном представительным. В поисковых запросах можно определять условия на значение или интервал значений атрибутов и связывать эти условия логическими операторами И/ИЛИ/КРОМЕ. Схема БД для СУБД *НИКА* создается автоматически в результате интеграции структур, определяемых формами, и корректируется при изменении старых или появлении новых форм.

В состав *МАГИС* входят также средства генерации ЯОД-описаний логических схем для баз данных стандарта *XBASE* (*dBASE*, *FoxBase*, *Clipper*).

Заключение

Созданные в стране CASE-средства значительно отличаются диапазоном охвата этапов ЖЦ и степенью оформления в виде рыночного изделия. Наибольшее число этапов поддержано инструментальными средствами *ИКС* и *СИНТЕЗ*. Надежно доведены до состояния промышленного использования и документированы *АРИУС*, *CASE.Аналитик* и *МАГИС*. Другие CASE-средства представляют собой экспериментальные или полупромышленные версии.

«Лидер И ресурсы» — в этой формуле рождения сложной системы, которой и является полномасштабный CASE-продукт, бескомпромиссное логическое И пока не дало в нашей стране значения TRUE. Однако интегральные усилия специалистов безусловно выводят отечественную CASE-технологию из состояния беспомощности перед функциональностью зарубежных CASE-средств. ■

Литература

1. Позин Б.А. CASE — новые технологии в информатизации общества// Проблемы информатизации.– 1992.– № 2.
2. Калянов Г.Н. Современные CASE-технологии. — М.: ИПУ. Препринт, 1992.
3. Штрик А.А. Зарубежные CASE средства создания программного обеспечения в отечественных разработках// Монитор.– 1993.– № 3.
4. Бойко В.В., Савинков В.М. Проектирование баз данных информационных систем.– М.: Финансы и статистика, 1989.
5. Ломако Е.И., Гуков Л.И., Морозова А.В. и др. Макетирование, проектирование и реализация диалоговых информационных систем.– М.: Финансы и статистика.– 1993.
6. Система автоматизации проектирования АСУ АРИУС. Вопросы построения и технологии применения.– Тула: НПО ОКА.– 1989.
7. Геловани В.А., Безруков Д.И., Бритков В.Б., Голосов А.О., Цаленко М.Ш. Интерактивная разработка информационных систем// Техническая кибернетика.– 1986.– № 2.
8. Бойко В.В., Цветков С.В., Глазов А.В. Дизайнер баз данных/ IV Всесоюзная конференция «Системы баз данных и знаний»: Тезисы докладов, Калинин, 1989.
9. Семейство САПР «Синтез»: Каталог организаций и программных средств, экспонируемых на международной выставке-ярмарке SOFTOOL-91. — М.: ВДНХ СССР, 1991.
10. Кабаков Ю.Б., Медведева А. И., Фурман Г.И. КОМОД-91 — система поддержки концептуальных схем и гипертекстов// УСиМ.– 1991.– № 7.
11. Алтухова А.Н., Чумаков Ю.С. СЛОВАРЬ ПРОЕКТА — CASE-средство для аналитиков и проектировщиков баз данных/ Семинар «CASE-технология»: Сб. материалов, Москва, 1992.
12. CASE.Аналитик: Рекламный буклет выставки SOFTOOL-92.– М.: Всероссийский выставочный центр, 1992.
13. Сивашев В.Л. Инструментальная система CASE*ДИАГРАММЕР как средство разработки и проектирования информационных систем/ Семинар «CASE-технология»: Сб. материалов, Москва, 1992.
14. А. Богачева, Н. Емельянов, А. Романов. Генерация информационных систем по формам входных и выходных документов// PC Magazine/ Russian Edition.– 1993. — № 1.

Алгоритмы сжатия информации

Д. Мастрюков

Часть 1. Сжатие по Хаффмену

Введение

Эта статья — первая в цикле, посвященном систематическому изложению алгоритмов сжатия информации. Почему эта серия статей может быть интересна для Вас?

Вы хотите знать, как работают такие популярные программы сжатия информации, как PKZIP, ARJ, LHA, STACKER, SuperStor? Те алгоритмы сжатия, которыми они пользуются, будут описаны подробно и систематично, разумеется, включая и исходные тексты работающих программ.

Вы хотите использовать сжатие информации в программах, которые Вы пишете? Вы сможете с легкостью заимствовать и модифицировать приведенные примеры. Если Вы хотите впоследствии использовать эти программы в коммерческих целях, то ссылки на владельцев патентов на приведенные алгоритмы помогут Вам не попасть в двусмысленное положение.

Вас интересует MultiMedia? Здесь Вы сможете найти ответ на вопрос, почему алгоритмы сжатия графических образов с потерями стали ключевым фактором, сделавшим возможной эту технологическую революцию. Примеры, реализующие часть известного стандарта JPEG, дадут Вам возможность самим экспериментировать с его возможностями.

Вас интересует научная сторона вопросов сжатия информации? Систематичность изложения и короткие математические выкладки помогут Вам получить достаточно основательные знания по этой проблеме.

ЧИТАЙТЕ ДАЛЕЕ

Содержание последующих статей

Сжатие без потерь

- Алгоритм Хаффмена. Адаптивное кодирование Хаффмена
- Арифметическое кодирование
- Алгоритмы группы LZ (Lempel-Ziv)
- Алгоритм LZW (Lempel-Ziv-Welch)
- Комбинированные схемы в PKZIP и LHA
- Алгоритмы STACKER и SuperStor

Сжатие с потерями

- Сжатие речевой информации
- Сжатие графических образов

Магическая формула: СЖАТИЕ = МОДЕЛИРОВАНИЕ + КОДИРОВАНИЕ

Вообще говоря, сжатие информации представляет собой процесс обработки символов некоторого сооб-

ТЕРМИНЫ

Терминологический словарь

Как и любая научная дисциплина, сжатие информации использует свой своеобразный лексикон, который может показаться несколько странным для новичка. Во избежание дальнейших недоразумений приведем основные понятия:

сжатие информации –
это процесс сокращения количества битов, необходимых для хранения некоторого объема информации;

сжатие без потерь –
информация, восстановленная из сжатого состояния, в точности соответствует исходной (до начала сжатия);

сжатие с потерями –
информация, восстановленная после сжатия, только частично соотвествует исходной (применяется при обработке изображений и звука).

щения и перевода этих символов в некоторые коды. Если этот процесс организован эффективно, то получающееся в результате закодированное сообщение занимает меньше места, чем исходное.

При просмотре обрабатываемого сообщения алгоритм сжатия реализует два почти независимых друг от друга процесса:

- поддерживает модель обрабатываемого сообщения;
- на основании модели кодирует очередной фрагмент сообщения;

Обычно весь процесс сжатия ошибочно отождествляется только с процессом кодирования, в то время как кодирование представляет собой только часть процесса сжатия, взаимодействующую с моделью данных. Используя один и тот же метод кодирования, можно совершенно по-разному строить модель сжимаемых данных. При этом результат сжатия будет отличаться для различных методов моделирования.

ТЕХНОЛОГИИ

Обычно сжатие отождествляется только с кодированием, в то время как кодирование представляет собой только часть этого процесса, взаимодействующую с моделью данных.

Так что, если Вы когда-либо услышите заявления типа: «Кодирование Хаффмена дает оптимальные результаты, это доказано математически», — или наоборот: «Кодирование Хаффмена вообще не дает хороших результатов», — отнеситесь к этому спокойно. И то и другое, по крайней мере, — некорректные утверждения. В каждом случае общие результаты работы алгоритма сжатия зависят и от метода моделирования и от метода кодирования.

ТЕРМИНЫ

Терминологический словарь

Приведенные определения не претендуют на безукоризненность, но помогут лучше понять модель данных, используемую при кодировании Хаффмена.

Граф —
совокупность множества узлов и множества дуг, направленных от одного узла к другому.

Дерево —
граф, обладающий следующими свойствами:
а) ни в один из узлов не входит более одной дуги (то есть отсутствуют циклы);
б) только в один узел не входит ни одной дуги, он называется корнем дерева;
в) перемещаясь по дугам от корня, можно попасть в любой узел.

Лист дерева —
узел, из которого не выходит ни одной дуги.
В паре узлов дерева, соединенных между собой дугой, тот, из которого она выходит, называется родителем, другой же — ребенком.
Два узла называются братьями, если имеют одного и того же родителя.

Двоичное дерево —
дерево, у которого из всех узлов, кроме листьев, выходит по две дуги.

Дерево кодирования Хаффмена (далее Н–дерево) —
двоичное дерево, у которого каждый узел имеет вес, и вес родителя равен суммарному весу его детей.

Входной алфавит —
множество символов, входящих в сообщение.

Кодирование Хаффмена

В конце сороковых годов, на заре развития теории информации, идеи разработки новых эффективных способов кодирования информации носились в воздухе. Исследователи занимались вопросами энтропии, содержимого информации и избыточности. Интересно, что эти первоначальные работы в области сжатия информации велись до появления современного цифрового компьютера. Сегодня теория информации развивается параллельно с программированием, но в то время идея разработки алгоритмов, использующих двоичную арифметику для кодирования символов, была значительным шагом вперед.

Один из первых алгоритмов эффективного кодирования информации был предложен Д.А.Хаффменом в 1952 году. Идея алгоритма состоит в следующем: зная вероятности вхождения символов в сообщение, можно описать процедуру построения кодов переменной длины, состоящих из целого количества битов. Символам с большей вероятностью присваиваются более короткие коды. Коды Хаффмена имеют уникальный префикс, что и позволяет однозначно их декодировать, несмотря на их переменную длину.

Классический алгоритм Хаффмена на входе получает таблицу частот встречаемости символов в сообщении. Далее на основании этой таблицы строится дерево кодирования Хаффмена (Н-дерево). Алгоритм построения Н-дерева прост и элегантен.

1. Символы входного алфавита образуют список свободных узлов. Каждый лист имеет вес, который может быть равен либо вероятности, либо количеству вхождений символа в сжимаемое сообщение.
2. Выбираются два свободных узла дерева с наименьшими весами.
3. Создается их родитель с весом, равным их суммарному весу.
4. Родитель добавляется в список свободных узлов, а двое его детей удаляются из этого списка.
5. Одной дуге, выходящей из родителя, ставится в соответствие бит 1, другой — бит 0.
6. Шаги, начиная со второго, повторяются до тех пор, пока в списке свободных узлов не останется только один свободный узел. Он и будет считаться корнем дерева.

Допустим, у нас есть следующая таблица частот:

15	7	6	6	5
А	Б	В	Г	Д

На первом шаге из листьев дерева выбираются два с наименьшими весами — Г и Д. Они присоединяются к новому узлу-родителю, вес которого устанавливается в 5 + 6 = 11. Затем узлы Г и Д удаляются из списка свободных. Узел Г соответствует ветви 0 родителя, узел Д — ветви 1.

На следующем шаге то же происходит с узлами Б и В, так как теперь эта пара имеет самый меньший вес в дереве. Создается новый узел с весом 13, а узлы Б и В удаляются из списка свободных. После всего этого дерево кодирования выглядит так, как показано на рис. 1.

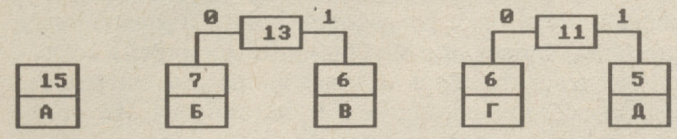


Рис. 1. Дерево кодирования Хаффмена после второго шага

На следующем шаге «наилегчайшей» парой оказываются узлы Б/В и Г/Д. Для них еще раз создается родитель, теперь уже с весом 24. Узел Б/В соответствует ветви 0 родителя, Г/Д — ветви 1.

На последнем шаге в списке свободных осталось только два узла — это А и узел (Б/В)/(Г/Д). В очередной раз создается родитель с весом 39 и бывшие свободными узлы присоединяются к разным его ветвям.

Поскольку свободным остался только один узел, то алгоритм построения дерева кодирования Хаффмена завершается. Н-дерево представлено на рис. 2.

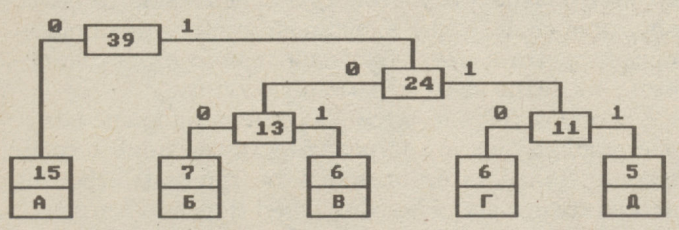


Рис. 2. Окончательное дерево кодирования Хаффмена

Чтобы определить код для каждого из символов, входящих в сообщение, мы должны пройти путь от листа дерева, соответствующего этому символу, до корня дерева, накапливая биты при перемещении по ветвям дерева. Полученная таким образом последовательность битов является кодом данного символа, записанным в обратном порядке.

Для данной таблицы символов коды Хаффмена будут выглядеть следующим образом.

А	0
Б	100
В	101
Г	110
Д	111

Поскольку ни один из полученных кодов не является префиксом другого, они могут быть однозначно декодированы при чтении их из потока. Кроме того, наиболее частый символ сообщения А закодирован наименьшим количеством битов, а наиболее редкий символ Д — наибольшим.

Классический алгоритм Хаффмена имеет один существенный недостаток. Для восстановления содержимого сжатого сообщения декодер должен знать таблицу частот, которой пользовался кодер. Следовательно, длина сжатого сообщения увеличивается на длину таблицы частот, которая должна посылаться впереди данных, что может свести на нет все усилия по сжатию сообщения. Кроме того, необходимость наличия полной частотной статистики перед началом собственно кодирования требует двух проходов по сообщению: одного для построения модели сообщения (таблицы частот и Н-дерева), другого для собственно кодирования.

Адаптивное сжатие

Адаптивное сжатие позволяет не передавать модель сообщения вместе с ним самим и ограничиться одним проходом по сообщению как при кодировании, так и при декодировании.

Практически любая форма кодирования может быть конвертирована в адаптивную. В общем случае программа, реализующая адаптивное сжатие, может быть выражена в следующей форме:

```
ИнициализироватьМодель();

Пока не конец сообщения

  Символ = ВзятьСледующийСимвол();
  Закодировать (Символ);
  ОбновитьМодельСимволом (Символ);

Конец Пока
```

Декодер в адаптивной схеме работает аналогичным образом:

```
ИнициализироватьМодель();

Пока не конец сжатой информации

  Символ = РаскодироватьСледующийСимвол();
  ВыдатьСимвол (Символ);
  ОбновитьМодельСимволом (Символ);

Конец Пока
```

Схема адаптивного кодирования/декодирования работает благодаря тому, что и при кодировании, и при декодировании используются одни и те же процедуры «ИнициализироватьМодель» и «ОбновитьМодельСимволом». И компрессор, и де-

компрессор начинают с «пустой» модели (не содержащей информации о сообщении) и с каждым просмотренным символом обновляют ее одинаковым образом.

Адаптивное кодирование Хаффмена

Следующим шагом в развитии алгоритма Хаффмена стала его адаптивная версия. Ей в основном и посвящена эта статья.

В создании алгоритма адаптивного кодирования Хаффмена наибольшие сложности возникают при разработке процедуры ОбновитьМодельСимволом(); можно было бы просто вставить внутрь этой процедуры полное построение дерева кодирования Хаффмена. В результате мы получили бы самый медленный в мире алгоритм сжатия, так как построение Н-дерева — это слишком большая работа и производить ее при обработке каждого символа неразумно. К счастью, существует способ модифицировать уже существующее Н-дерево так, чтобы отобразить обработку нового символа.

Упорядоченное дерево

Будем говорить, что дерево обладает свойством упорядоченности, если его узлы могут быть перечислены в порядке возрастания веса и в этом перечислении каждый узел находится рядом со своим братом. Пример упорядоченного дерева приведен на рис. 3. Здесь W — вес узла, N — порядковый номер в списке узлов.

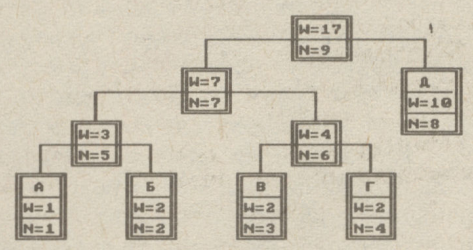


Рис. 3. Свойство упорядоченности дерева кодирования Хаффмена

Примем без доказательства утверждение о том, что двоичное дерево является деревом кодирования Хаффмена тогда и только тогда, когда оно удовлетворяет свойству упорядоченности.

Сохранение свойства упорядоченности в процессе обновления дерева позволяет нам быть уверенными в том, что двоичное дерево, с которым мы работаем, — это Н-дерево и до, и после обновления веса у листьев дерева.

Обновление дерева при считывании очередного символа сообщения состоит из двух операций.

Первая — увеличение веса узлов дерева — представлена на рис. 4. Вначале увеличиваем вес листа, соответствующего считанному символу, на единицу. Затем увеличиваем вес родителя, чтобы привести его в соответствие с новыми значениями веса у детей. Этот процесс продолжается до тех пор, пока мы не доберемся до корня дерева. Среднее число операций увеличения веса равно среднему количеству битов, необходимых для того, чтобы закодировать символ.

Вторая операция — перестановка узлов дерева — требуется тогда, когда увеличение веса узла приводит

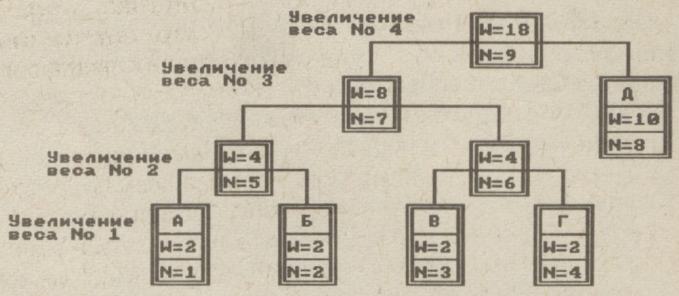


Рис. 4. Процедура обновления дерева кодирования при увеличении веса листа А

к нарушению свойства упорядоченности, то есть тогда, когда увеличенный вес узла стал больше, чем вес следующего по порядку узла (рис. 5). Если и дальше продолжать обрабатывать увеличение веса, двигаясь к корню дерева, то наше дерево перестанет быть деревом Хаффмена.

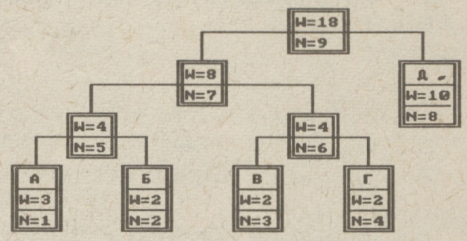


Рис. 5. При увеличении веса листа А нарушается свойство упорядоченности

Чтобы сохранить упорядоченность дерева кодирования, алгоритм работает следующим образом. Пусть новый увеличенный вес узла равен W+1. Тогда начинаем двигаться по списку в сторону увеличения веса, пока не найдем последний узел с весом W. Переставим текущий и найденный узлы между собой в списке (рис. 6), восстанавливая таким образом порядок в дереве. (При этом родители каждого из узлов тоже изменятся.) На этом операция перестановки заканчивается.

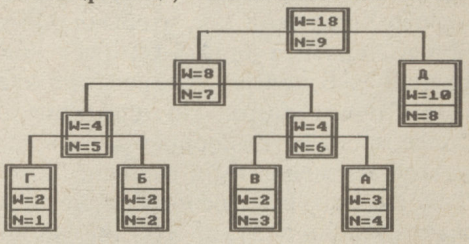


Рис. 6. Дерево кодирования после первой перестановки узлов (Г и А)

После перестановки операция увеличения веса узлов продолжается дальше. Следующий узел, вес которого будет увеличен алгоритмом, — это новый родитель узла, увеличение веса которого вызвало перестановку.

Предположим, что символ А встретился в сообщении еще два раза подряд. Дерево кодирования после двукратного вызова процедуры обновления показано на рис. 7.

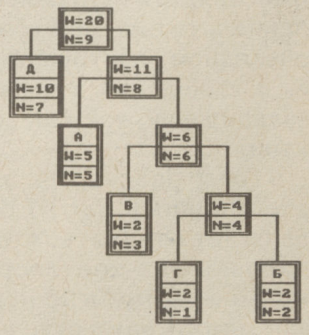


Рис. 7. Дерево после завершения процедуры обновления

Обратите внимание, как обновление дерева кодирования отражается на длине кодов Хаффмена для входящих в данное сообщение символов.

В целом алгоритм обновления дерева может быть записан следующим образом:

```
ОбновитьМодельСимволом (Символ)
{
  ТекущийУзел = ЛистСоответствующий (Символ)
  Всегда
     УвеличитьВес (ТекущийУзел)
     Если ТекущийУзел = КореньДерева
        Выход;
     Если   Вес (ТекущийУзел) >
            Вес (СледующийЗа (ТекущийУзел))
        Перестановка ();
     ТекущийУзел = Родитель (ТекущийУзел);
  Конец Всегда
}
```

Проблемы адаптивного кодирования

Инициализация

Хорошо было бы, чтобы кодер не тратил зря кодовое пространство на символы, которые не встречаются в сообщении.

Если речь идет о классическом алгоритме Хаффмена, то те символы, которые не встречаются в сообщении, уже известны до начала кодирования, так как известна таблица частот и символы, у которых частота встречаемости равна 0. В адаптивной версии алгоритма мы не можем знать заранее, какие символы появятся в сообщении. Можно проинициализировать дерево Хаффмена так, чтобы оно имело все 256 символов алфавита (для 8-битовых кодов) с частотой, равной 1. В начале кодирования каждый код будет иметь длину 8 битов. По мере адаптации модели наиболее часто встречающиеся символы будут кодироваться все меньшим и меньшим количеством битов. Такой подход работоспособен, но он значительно снижает степень сжатия, особенно на коротких сообщениях.

Лучше начинать моделирование с пустого дерева и добавлять в него символы только по мере их появления в сжимаемом сообщении. Но это приводит к очевидному противоречию: когда символ появляется в сообщении в первый раз, он не может быть закодирован, так как его еще нет в дереве кодирования.

Чтобы разрешить это противоречие, введем специальный ESCAPE код, который для декодера будет означать, что следующий символ закодирован вне контекста модели сообщения. Например, его можно передать в поток сжатой информации как есть, не кодируя вообще. Метод «ЗакодироватьСимвол» в алгоритме адаптивного кодирования Хаффмена можно записать следующим образом.

```
ЗакодироватьСимвол (Символ)
{
  Если СимволУжеЕстьВТаблице (Символ)
     ВыдатьКодХаффменаДляСимвола (Символ)
  Иначе
  {
     ВыдатьКодХаффменаДляСимвола (ESCAPE)
     ВыдатьСимвол (Символ)
  }
}
```

Использование специального символа ESCAPE подразумевает определенную инициализацию дерева до начала кодирования и декодирования: в него помещаются 2 специальных символа: ESCAPE и EOF (конец файла), с весом, равным 1 (рис. 8). Поскольку процесс обновления дерева не коснется их веса, то по ходу кодирования они будут перемещаться на самые удаленные ветви дерева и иметь самые длинные коды.

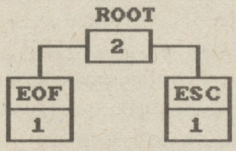


Рис. 8. Дерево кодирования после инициализации

Переполнение

В процессе работы алгоритма сжатия вес узлов в дереве кодирования Хаффмена неуклонно растет. Первая проблема возникает тогда, когда вес корня дерева начинает превосходить вместимость ячейки, в которой он хранится. Как правило, это 16-битовое значение и, следовательно, не может быть больше, чем 65535. Вторая проблема, заслуживающая еще большего внимания, может возникнуть значительно раньше, когда размер самого длинного кода Хаффмена превосходит вместимость ячейки, которая используется для того, чтобы передать его в выходной поток. Декодеру все равно, какой длины код он декодирует, поскольку он движется сверху вниз по дереву кодирования, выбирая из входного потока по одному биту. Кодер же должен начинать от листа дерева и двигаться вверх к корню, собирая биты, которые нужно передать. Обычно это происходит с переменной типа «целое», и, когда длина кода Хаффмена превосходит размер типа «целое» в битах, наступает переполнение.

Можно доказать, что максимальную длину код Хаффмена для сообщений с одним и тем же входным алфавитом будет иметь, если частоты символов образует последовательность Фибоначчи (рис. 9).

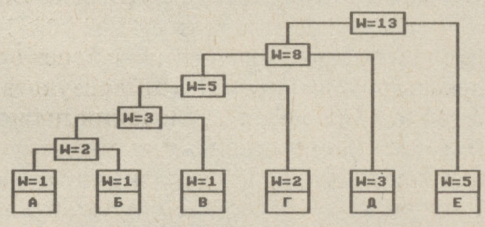


Рис. 9. Дерево Хаффмена на числах Фибоначчи

Функция Фибоначчи определяется следующим образом.

```
int Fib (int n)
{
  if (n <= 1)
      return 1;
  else
      return (fib (n — 1) + fib (n — 2));
}
```

Если вес корневого узла в дереве Хаффмена равен Fib (i), то наиболее длинный код, возможный для этого дерева, имеет длину i-1.

Это означает, что если «целые», используемые для представления весов в дереве Хаффмена, имеют длину 16 битов, то вес корня, равный 4181 (Fib(18)), может привести к переполнению. (Вообще говоря, сообщение с частотами символов, равными числам Фибоначчи до Fib(18), — это отличный способ протестировать работу программы сжатия по Хаффмену.)

Масштабирование весов узлов дерева Хаффмена

Принимая во внимание сказанное выше, алгоритм обновления дерева Хаффмена должен быть изменен следующим образом: при увеличении веса нужно проверять его на достижение допустимого максимума. Если мы достигли максимума, то необходимо «масштабировать» вес, обычно разделив вес листьев на целое число, например, 2, а потом пересчитав вес всех остальных узлов.

Однако при делении веса пополам возникает проблема, связанная с тем, что после выполнения этой операции дерево может изменить свою форму. Объясняется это тем, что мы делим целые числа и при делении отбрасываем дробную часть (рис. 10-12).

Правильно организованное дерево Хаффмена после масштабирования может иметь форму, значительно отличающуюся от исходной. Это происходит потому, что масштабирование приводит к потере точности нашей статистики. Но со сбором новой статистики последствия этих «ошибок» практически сходят на нет. Масштабирование веса — довольно дорогостоящая операция, так как она приводит к необходимости заново строить все дерево кодирования. Но, так как необходимость в ней возникает относительно редко, то с этим можно смириться.

Выигрыш от масштабирования

Масштабирование веса узлов дерева через определенные интервалы дает неожиданный результат. Несмотря на то, что при масштабировании происходит потеря точности статистики, тесты показывают, что оно приводит к лучшим показателям сжатия, чем если бы масштабирование откладывалось. Это можно объяснить тем, что текущие символы сжимаемого потока больше «похожи» на своих близких предшественников, чем на тех, которые встречались намного раньше. Масштабирование приводит к уменьшению влияния «давних» символов на статистику и к увеличению влияния на нее «недавних» символов. Это очень сложно измерить количественно, но, в принципе, масштабирование оказывает положительное влияние на степень сжатия информации. Эксперименты с масштабированием в различных точках процесса сжатия показывают, что степень сжатия сильно зависит от момента масштабирования веса, но не существует правила выбора оптимального момента масштабирования для программы, ориентированной на сжатие любых типов информации.

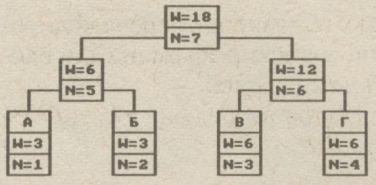


Рис. 10. Дерево кодирования перед масштабированием весов

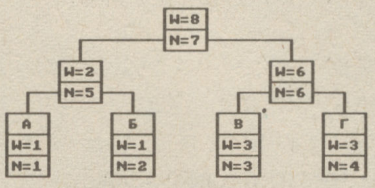


Рис. 11. Дерево кодирования после деления веса каждого листа на 2. Свойство упорядоченности

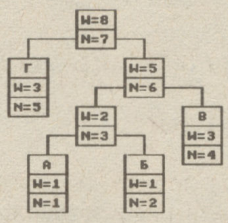


Рис. 12. Правильная форма дерева после масштабирования

Заключение

С тех пор, как Д.А.Хаффмен опубликовал в 1952 году свою работу «Метод построения кодов с минимальной избыточностью», его алгоритм кодирования стал базой для огромного количества дальнейших исследований в этой области. По сей день в компьютерных журналах можно найти большое количество публикаций, посвященных как различным реализациям алгоритма Хаффмена, так и поискам его лучшего применения. Кодирование Хаффмена используется в коммерческих программах сжатия (например, в PKZIP и LHA), встроено в некоторые телефаксы и даже используется в алгоритме JPEG сжатия графических изображений с потерями.

Приведенная далее программа, реализующая алгоритм адаптивного кодирования Хаффмена, написана на языке С и может быть собрана компиляторами фирм Borland, Microsoft и Zortech. Она не была сколько-нибудь оптимизирована по скорости и дает не сравнимые с PKZIP, например, результаты по сжатию (что абсолютно закономерно, так как PKZIP использует совершенно другой способ моделирования данных для кодирования Хаффмена). Автор надеется, что приведенные комментарии, а также возможность «поиграть» с исходным текстом в отладчике или

профайлере помогут Вам получить достаточно полное представление о том, как работает схема сжатия Хаффмена. ■

ЧИТАЙТЕ ДАЛЕЕ

В следующем номере

Следующая статья этой серии будет посвящена арифметическому кодированию. Этот алгоритм является таким же краеугольным камнем в сжатии информации, как и алгоритм Хаффмена, в отличие от которого арифметическое кодирование позволяет превратить один байт входного потока в менее, чем один бит кода (в некоторых случаях). О том, как это делается, читайте в следующем номере «Монитора».

Рекомендуемая литература

1. Huffman D.A. A Method for the construction of minimum-redundancy codes// IRE.– September 1952.– Vol. 40.– № 9. Русский перевод: Д.А. Хаффмен. Метод построения кодов с минимальной избыточностью// Кибернетический сборник.– 1961.– Вып. 3.
2. Storer J.A. Data compression// Computer Science Press.– 1988.
3. Nelson M. The Data Compression Book.– M&T Publishing.– 1991.
4. Gallager R.G. Variations on a theme by Huffman// IEEE Transactions on Information Theory.– 1978.– Vol. 24.– № 6.
5. Knuth D.E. Dynamic Huffman Coding// Journal of Algorithms.– 1985.– № 6.
6. Gormak G.V and Horspool R.N. Algorithms for adaptive Huffman codes// Information Processing Letters.– 1984.– № 18.
7. Jacobsson M. Huffman coding in Bit-Vector Compression// Information Processing Letters.– 1978.– Vol. 7.– № 6.
8. Johnsen O. On the redundancy of binary Huffman codes// IEEE Transactions on Information Theory.– 1980.– Vol. 26.– № 2.
9. D.R. McIntyre and M.A. Pechura. Data Compression Using Static Huffman Code-Decode Tables// Journal of the ACM.– 1985.– Vol. 28.– № 6.
10. Д. Арапов. Пишем упаковщик// Монитор.– 1993.– № 1.

Русскоязычная версия!

Just Write™

SYMANTEC

Диалог-МИФИ
095-320-3466

Merisel-Russia
095-276-9008

Параграф
095-299-7923

Перспективные технологии
095-256-6271

Трио Плюс
095-971-1204

Прекрасный текстовый процессор...
Для Windows™...
Для тех, кто хочет быстро, просто и красиво работать с текстом...

Листинг

```
/*
* Адаптивное кодирование Хаффмена
* Включаемый файл
*/
/*
* Константы, используемые
* в алгоритме кодирования
*/
#define END_OF_STREAM   256
/* Маркер конца потока */
#define ESCAPE          257
/* Маркер начала ESCAPE последовательности */
#define SYMBOL_COUNT    258
/* Максимально возможное количество
* листьев дерева (256+2 маркера) */
#define NODE_TABLE_COUNT ((SYMBOL_COUNT*2)-1)
#define ROOT_NODE       0
#define MAX_WEIGHT      0x8000
/* Вес корня, при котором начинается
* масштабирование веса */
#define TRUE            1
#define FALSE           0
/*
* Эта структура данных нужна
* для побитового доступа к файлам
*/
typedef struct bit_file
{
    FILE      *file;
    unsigned char mask;
    int       rack;
    int       pacifier_counter;
}
COMPRESSED_FILE;
/*
* Эта структура данных используется
* для работы с деревом кодирования
* Хаффмена процедурами кодирования
* и декодирования
*/
typedef struct tree
{
    int leaf[ SYMBOL_COUNT ];
    /* Массив листьев дерева */
    int next_free_node;
    /* Номер следующего свободного
    элемента массива листьев */
    struct node
    {
        unsigned int weight;
        /* Вес узла */
        int parent;
        /* Номер родителя в массиве узлов */
        int child_is_leaf;
        /* Флаг листа (TRUE, если лист) */
        int child;
    }
    nodes[ NODE_TABLE_COUNT ];
    /* Массив узлов */
}
TREE;
extern char *Usage;
extern char *CompressionName;
/*
* Сервисные функции
*/
void usage_exit    (char *prog_name);
void print_ratios (char *input,
                   char *output);
long file_size     (char *name);
void fatal_error   (char *fmt, ...);
/*
* Функции побитового доступа к файлам
*/
COMPRESSED_FILE    *OpenInputCompressedFile
                    (char *name);
COMPRESSED_FILE    *OpenOutputCompressedFile
                    (char *name);
void OutputBit     (COMPRESSED_FILE *, int bit);
void OutputBits    (COMPRESSED_FILE *bit_file,
                    unsigned long code,
                    int count);
int InputBit       (COMPRESSED_FILE *bit_file);
unsigned long      InputBits    (COMPRESSED_FILE
                                 *bit_file,
                                 int bit_count);
void CloseInputCompressedFile
                (COMPRESSED_FILE bit_file);
void CloseOutputCompressedFile
                (COMPRESSED_FILE bit_file);
/*
* Собственно адаптивное кодирование Хаффмена
*/
void CompressFile    (FILE *input,
                      COMPRESSED_FILE *output);
void ExpandFile      (COMPRESSED_FILE *input,
                      FILE *output);
void InitializeTree  (TREE *tree);
void EncodeSymbol    (TREE *tree,
                      unsigned int c,
                      COMPRESSED_FILE *output);
int DecodeSymbol     (TREE *tree,
                      COMPRESSED_FILE *input);
void UpdateModel     (TREE *tree, int c);
void RebuildTree     (TREE *tree);
void swap_nodes      (TREE *tree, int i, int j);
void add_new_node    (TREE *tree, int c);
/*
* Конец включаемого файла
*/
```

```
/*
* Адаптивное кодирование Хаффмена:
* демонстрационная программа
*
* Компиляция:
*  Borland C   - bcc huffman.c
*  Zortech C   - ztc huffman.c
*  Microsoft C - cl  huffman.c
* Использование:
*  huffman input_file output_file
*/
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <ctype.h>
#include <stdarg.h>
#include "huffman.h"
/*
* Функция Main осуществляет разбор
* командной строки и проверку при
* открытии входного и выходного файла.
* Затем запускает процедуру сжатия
* и печатает результаты работы.
* ВНИМАНИЕ: приведенная программа
* может только сжимать файлы.
*/
int main (int argc, char *argv[])
{
    COMPRESSED_FILE *output;
    FILE *input;
    setbuf(stdout, NULL);
    if (argc < 3)
        usage_exit(argv[ 0 ]);
    input = fopen(argv[ 1 ], "rb");
    if (input == NULL)
        fatal_error("Не могу открыть файл %s
                    для чтения\n", argv[ 1 ]);
    output = OpenOutputCompressedFile(argv[ 2 ]);
    if (output == NULL)
        fatal_error("Не могу открыть выходной
                    файл %s\n", argv[ 2 ]);
    printf("\nСжимаю файл %s в %s.\n",
            argv[ 1 ], argv[ 2 ]);
    printf("используя %s\n", CompressionName);
    CompressFile(input, output);
    CloseOutputCompressedFile(output);
    fclose(input);
    print_ratios(argv[ 1 ], argv[ 2 ]);
    return(0);
}
/*
* Эта процедура вызывается при попытке
* запустить программу без
* параметров. Она печатает сообщения
* о правильном использовании программы.
*/
void usage_exit (char *prog_name)
{
    char *short_name;
    char *extension;
    short_name = strrchr(prog_name, '\\');
    if (short_name == NULL)
        short_name = strrchr(prog_name, '/');
    if (short_name == NULL)
        short_name = strrchr(prog_name, ':');
    if (short_name != NULL)
        short_name++;
    else
        short_name = prog_name;
    extension = strrchr(short_name, '.');
    if (extension != NULL)
        *extension = '\0';
    printf("\nИспользование: %s %s\n",
            short_name, Usage);
    exit(0);
}
/*
* Эта функция возвращает размер
* указанного ей файла
```

```
*/
#ifndef SEEK_END
#define SEEK_END 2
#endif
long file_size (char *name)
{
    long eof_ftell;
    FILE *file;
    file = fopen(name, "r");
    if (file == NULL)
        return(0L);
    fseek(file, 0L, SEEK_END);
    eof_ftell = ftell(file);
    fclose(file);
    return(eof_ftell);
}
/*
* Эта процедура печатает результаты сжатия
* после окончания работы процедуры сжатия
*/
void print_ratios (char *input, char *output)
{
    long input_size;
    long output_size;
    int ratio;
    input_size = file_size(input);
    if (input_size == 0)
        input_size = 1;
    output_size = file_size(output);
    ratio =  100 - (int)
             (output_size * 100L / input_size);
    printf("\nРазмер несжатого файла:\t%ld\n",
           input_size);
    printf("Размер сжатого файла:\t%ld\n",
           output_size);
    if (output_size == 0)
        output_size = 1;
    printf("Степень сжатия:\t\t%d%%\n", ratio);
}
/*
* Эта процедура вызывается при фатальных
* ошибках при выполнении программы
*/
void fatal_error (char *fmt, ...)
{
    va_list argptr;
    va_start(argptr, fmt);
    printf("Фатальная ошибка: ");
    vprintf(fmt, argptr);
    va_end(argptr);
    exit(-1);
}
/*
* Эта константа используется как шаг
* индикатора выполнения
*/
#define PACIFIER_COUNT 2047
/*
* Открытие файла для побитового вывода
*/
COMPRESSED_FILE   *OpenOutputCompressedFile
                      (char *name)
{
    COMPRESSED_FILE *compressed_file;
    compressed_file = (COMPRESSED_FILE *)
                      calloc(1,
                      sizeof(COMPRESSED_FILE));
    if (compressed_file == NULL)
        return(compressed_file);
    compressed_file->file = fopen(name, "wb");
    compressed_file->rack = 0;
    compressed_file->mask = 0x80;
    compressed_file->pacifier_counter = 0;
    return(compressed_file);
}
/*
* Открытие файла для побитового ввода
*/
COMPRESSED_FILE   *OpenInputCompressedFile
                      (char *name)
{
    COMPRESSED_FILE *compressed_file;
    compressed_file = (COMPRESSED_FILE *)
                      calloc(1,
                      sizeof(COMPRESSED_FILE));
    if (compressed_file == NULL)
        return(compressed_file);
    compressed_file->file = fopen(name, "rb");
    compressed_file->rack = 0;
    compressed_file->mask = 0x80;
    compressed_file->pacifier_counter = 0;
    return(compressed_file);
}
/*
* Закрытие файла для побитового вывода
*/
void CloseOutputCompressedFile
        (COMPRESSED_FILE *compressed_file)
{
    if (compressed_file->mask != 0x80)
        if (putc(compressed_file->rack,
                 compressed_file->file) !=
                 compressed_file->rack)
            fatal_error("Фатальная ошибка
                         при попытке закрыть
                         сжатый файл!\n");
    fclose(compressed_file->file);
    free((char *) compressed_file);
}
/*
* Закрытие файла для побитового ввода
*/
void CloseInputCompressedFile
        (COMPRESSED_FILE *compressed_file)
{
    fclose(compressed_file->file);
    free((char *) compressed_file);
}
*
* Вывод одного бита
*/
void OutputBit
        (COMPRESSED_FILE *compressed_file,
        int bit)
{
    if (bit)
        compressed_file->rack |=
            compressed_file->mask;
    compressed_file->mask >>= 1;
    if (compressed_file->mask == 0)
    {
        if (putc(compressed_file->rack,
            compressed_file->file) !=
                compressed_file->rack)
            fatal_error("Фатальная ошибка
                         в процедуре OutputBit!\n");
        else if
        ((compressed_file->pacifier_counter++
        & PACIFIER_COUNT) == 0)
            putc('.', stdout);
        compressed_file->rack = 0;
        compressed_file->mask = 0x80;
    }
}
/*
* Вывод нескольких битов
*/
void OutputBits
        (COMPRESSED_FILE *compressed_file,
        unsigned long code, int count)
{
    unsigned long mask;
    mask = 1L << (count - 1);
    while (mask != 0)
    {
        if (mask & code)
            compressed_file->rack |=
            compressed_file->mask;
        compressed_file->mask >>= 1;
        if (compressed_file->mask == 0)
        {
            if (putc(compressed_file->rack,
                compressed_file->file) !=
                    compressed_file->rack)
                fatal_error("Фатальная ошибка
                             в процедуре OutputBits!\n");
            else if
                    ((compressed_file->
                    pacifier_counter++ &
                    PACIFIER_COUNT) == 0)
                putc('.', stdout);
            compressed_file->rack = 0;
            compressed_file->mask = 0x80;
        }
        mask >>= 1;
    }
}
/*
* Ввод одного бита
*/
int InputBit
        (COMPRESSED_FILE *compressed_file)
{
    int value;
    if (compressed_file->mask == 0x80)
    {
        compressed_file->rack =
        getc(compressed_file->file);
        if (compressed_file->rack == EOF)
```

```
            fatal_error("Фатальная ошибка
                в процедуре InputBit!\n");
        if ((compressed_file->pacifier_counter++
            & PACIFIER_COUNT) == 0)
            putc('.', stdout);
    }
    value =  compressed_file->rack &
             compressed_file->mask;
    compressed_file->mask >>= 1;
    if (compressed_file->mask == 0)
        compressed_file->mask = 0x80;
    return(value ? 1 : 0);
}
/*
* Ввод нескольких битов
*/
unsigned long InputBits
            (COMPRESSED_FILE *compressed_file,
             int bit_count)
{
    unsigned long mask;
    unsigned long return_value;
    mask = 1L << (bit_count - 1);
    return_value = 0;
    while (mask != 0)
    {
        if (compressed_file->mask == 0x80)
        {
            compressed_file->rack =
                getc(compressed_file->file);
            if (compressed_file->rack == EOF)
                fatal_error("Фатальная ошибка
                    в процедуре InputBits!\n");
            if ((compressed_file->
                 pacifier_counter++ &
                 PACIFIER_COUNT) == 0)
                putc('.', stdout);
        }
        if    (compressed_file->rack &
               compressed_file->mask)
            return_value |= mask;
        mask >>= 1;
        compressed_file->mask >>= 1;
        if (compressed_file->mask == 0)
            compressed_file->mask = 0x80;
    }
    return (return_value);
}
/*
* С этого места начинается исходный текст,
* реализующий собственно
* алгоритм адаптивного кодирования Хаффмена
*/
char *CompressionName   =
    "Адаптивное Кодирование Хаффмена";
char *Usage         =
    "что_сжимать куда_сжимать";
/*
* Дерево адаптивного кодирования Хаффмена
*/
TREE Tree;
/*
* Процедура преобразования входного файла
* в выходной сжатый файл
*/
void CompressFile(FILE *input,
COMPRESSED_FILE *output)
{
    int c;
    InitializeTree(&Tree);
    while ((c = getc(input)) != EOF)
    {
        EncodeSymbol(&Tree, c, output);
        UpdateModel(&Tree, c);
    }
    EncodeSymbol(&Tree, END_OF_STREAM, output);
}
/*
* Процедура декомпрессии упакованного файла
*/
void ExpandFile    (COMPRESSED_FILE *input,
                    FILE *output)
{
    int c;
    InitializeTree(&Tree);
    while ((c = DecodeSymbol(&Tree, input)) !=
           END_OF_STREAM)
    {
        if (putc(c, output) == EOF)
            fatal_error("Не могу писать в
                выходной файл при распаковке");
        UpdateModel(&Tree, c);
    }
}
/*
* Перед началом работы алгоритма дерево
* кодирования инициализируется
* двумя специальными (не ASCII) символами:
* ESCAPE и END_OF_STREAM.
* Так же инициализируется корень дерева.
* Все листья инициализируются -1, так как
* они еще не присутствуют в
* дереве кодирования.
*/
void InitializeTree (TREE *tree)
{
    int i;
    tree->nodes[ ROOT_NODE ].
        child           = ROOT_NODE + 1;
    tree->nodes[ ROOT_NODE ].
        child_is_leaf   = FALSE;
    tree->nodes[ ROOT_NODE ].
        weight          = 2;
    tree->nodes[ ROOT_NODE ].
        parent          = -1;
    tree->nodes[ ROOT_NODE + 1 ].
        child     = END_OF_STREAM;
    tree->nodes[ ROOT_NODE + 1 ].
        child_is_leaf   = TRUE;
    tree->nodes[ ROOT_NODE + 1 ].
        weight          = 1;
    tree->nodes[ ROOT_NODE + 1 ].
        parent          = ROOT_NODE;
    tree->leaf[ END_OF_STREAM ]
                        = ROOT_NODE + 1;
    tree->nodes[ ROOT_NODE + 2 ].
        child           = ESCAPE;
    tree->nodes[ ROOT_NODE + 2 ].
        child_is_leaf   = TRUE;
    tree->nodes[ ROOT_NODE + 2 ]
        .weight         = 1;
    tree->nodes[ ROOT_NODE + 2 ].
        parent          = ROOT_NODE;
    tree->leaf[ ESCAPE ] = ROOT_NODE + 2;
    tree->next_free_node = ROOT_NODE + 3;
    for (i = 0 ; i < END_OF_STREAM ; i++)
        tree->leaf[ i ] = -1;
}
/*
* Эта процедура преобразует входной символ в
* последовательность битов на основе
* текущего состояния дерева кодирования.
* Некоторое неудобство состоит в том, что,
* обходя дерево от листа к корню, мы
* получаем последовательность битов в
* обратном порядке, и поэтому необходимо
* аккумулировать биты в INTEGER переменной
* и выдавать их после того,
* как обход дерева закончен.
*/
void EncodeSymbol (TREE *tree,
                   unsigned int c,
                   COMPRESSED_FILE *output)
{
    unsigned long code;
    unsigned long current_bit;
    int code_size;
    int current_node;
    code = 0;
    current_bit = 1;
    code_size = 0;
    current_node = tree->leaf[ c ];
    if (current_node == -1)
        current_node = tree->leaf[ ESCAPE ];
    while (current_node != ROOT_NODE)
    {
        if ((current_node & 1) == 0)
            code |= current_bit;
        current_bit <<= 1;
        code_size++;
        current_node = tree->nodes
                       [ current_node ].parent;
    }
    OutputBits(output, code, code_size);
    if (tree->leaf[ c ] == -1)
    {
        OutputBits(output, (unsigned long) c, 8);
        add_new_node(tree, c);
    }
}
/*
* Процедура декодирования очень проста.
* Начиная от корня, мы обходим дерево,
* пока не дойдем до листа. Затем проверяем
* не прочитали ли мы ESCAPE код. Если да,
* то следующие 8 битов соответствуют
* незакодированному символу, который немедленно
* считывается и добавляется к таблице.
*/
int DecodeSymbol  (TREE *tree,
```

```
                        COMPRESSED_FILE *input)
{
    int current_node;
    int c;
    current_node = ROOT_NODE;
    while (!tree->nodes
            [ current_node ].child_is_leaf)
    {
        current_node = tree->nodes
            [ current_node ].child;
        current_node += InputBit(input);
    }
    c = tree->nodes[ current_node ].child;
    if (c == ESCAPE)
    {
        c = (int) InputBits(input, 8);
        add_new_node(tree, c);
    }
    return(c);
}
/*
* Процедура обновления модели кодирования
* для данного символа, пожалуй, самое
* сложное в адаптивном кодировании Хаффмена.
* См. соответствующую часть статьи.
*/
void UpdateModel (TREE *tree, int c)
{
    int current_node;
    int new_node;
    if (tree->nodes[ ROOT_NODE].weight ==
        MAX_WEIGHT)
        RebuildTree(tree);
    current_node = tree->leaf[ c ];
    while (current_node != -1)
    {
        tree->nodes[ current_node ].weight++;
        for    (new_node = current_node ;
                new_node > ROOT_NODE ; new_node--)
            if (tree->nodes
                    [ new_node - 1 ].weight >=
                    tree->nodes
                        [ current_node ].weight)
                break;
        if (current_node != new_node)
        {
            swap_nodes(tree, current_node,
                    new_node);
            current_node = new_node;
        }
        current_node = tree->nodes
                        [ current_node ].parent;
    }
}
/*
* Процедура перестроения дерева вызывается
* тогда, когда вес корня дерева достигает
* пороговой величины. Она начинается
* с простого деления весов узлов на 2.
* Но из-за ошибок округления при этом может
* быть нарушено свойство упорядоченности
* дерева кодирования,
* и необходимы дополнительные усилия,
* чтобы привести его в корректное состояние.
*/
void RebuildTree (TREE *tree)
{
    int i;
    int j;
    int k;
    unsigned int weight;
    printf("R");
    j = tree->next_free_node - 1;
    for (i = j ; i >= ROOT_NODE ; i--)
    {
        if (tree->nodes[ i ].child_is_leaf)
        {
            tree->nodes[ j ] = tree->nodes[ i ];
            tree->nodes[ j ].weight =
                (tree->nodes[ j ].weight + 1) / 2;
            j--;
        }
    }
    for    (i = tree->next_free_node - 2 ;
            j >= ROOT_NODE ; i -= 2, j--)
    {
        k = i + 1;
        tree->nodes[ j ].weight =
            tree->nodes[ i ].weight +
            tree->nodes[ k ].weight;
        weight = tree->nodes[ j ].weight;
        tree->nodes[ j ].child_is_leaf = FALSE;
        for    (k = j + 1 ;
                weight < tree->nodes[ k ].weight ;
                k++);
        k--;
        memmove(&tree->nodes[ j ],
                &tree->nodes[ j + 1 ],
                (k - j) * sizeof(struct node));
        tree->nodes[ k ].weight = weight;
        tree->nodes[ k ].child = i;
        tree->nodes[ k ].child_is_leaf = FALSE;
    }
    for    (i = tree->next_free_node - 1 ;
            i >= ROOT_NODE ; i--)
    {
        if (tree->nodes[ i ].child_is_leaf)
        {
            k = tree->nodes[ i ].child;
            tree->leaf[ k ] = i;
        }
        else
        {
            k = tree->nodes[ i ].child;
            tree->nodes[ k ].parent =
                tree->nodes[ k + 1 ].parent = i;
        }
    }
}
/*
* Процедура перестановки узлов дерева
* вызывается тогда, когда очередное
* увеличение веса узла привело к нарушению
* свойства упорядоченности.
*/
void swap_nodes (TREE *tree, int i, int j)
{
    struct node temp;
    if (tree->nodes[ i ].child_is_leaf)
        tree->leaf[ tree->nodes[ i ].child ] = j;
    else
    {
        tree->nodes[ tree->nodes[ i ].
            child ].parent = j;
        tree->nodes[ tree->nodes[ i ].
            child + 1 ].parent = j;
    }
    if (tree->nodes[ j ].child_is_leaf)
        tree->leaf[ tree->nodes[ j ].child ] = i;
    else
    {
        tree->nodes[ tree->nodes[ j ].
            child ].parent = i;
        tree->nodes[ tree->nodes[ j ].
            child + 1 ].parent = i;
    }
    temp = tree->nodes[ i ];
    tree->nodes[ i ] = tree->nodes[ j ];
    tree->nodes[ i ].parent = temp.parent;
    temp.parent = tree->nodes[ j ].parent;
    tree->nodes[ j ] = temp;
}
/*
* Добавление нового узла в дерево
* осуществляется достаточно просто.
* Для этого "самый легкий" узел дерева
* разбивается на 2, один из которых и есть
* тот новый узел. Новому узлу присваивается
* вес 0, который будет изменен потом, при
* нормальном процессе обновления дерева.
*/
void add_new_node (TREE *tree, int c)
{
    int lightest_node;
    int new_node;
    int zero_weight_node;
    lightest_node = tree->next_free_node - 1;
    new_node = tree->next_free_node;
    zero_weight_node = tree->next_free_node + 1;
    tree->next_free_node += 2;
    tree->nodes[ new_node ] =
                    tree->nodes[ lightest_node ];
    tree->nodes[ new_node ].parent
                    = lightest_node;
    tree->leaf[ tree->nodes[ new_node ].child ]
                    = new_node;
    tree->nodes[ lightest_node ].child
                    = new_node;
    tree->nodes[ lightest_node ].child_is_leaf
                    = FALSE;
    tree->nodes[ zero_weight_node ].child
                    = c;
    tree->nodes[ zero_weight_node ].child_is_leaf
                    = TRUE;
    tree->nodes[ zero_weight_node ].weight
                    = 0;
    tree->nodes[ zero_weight_node ].parent
                    = lightest_node;
    tree->leaf[ c ]   = zero_weight_node;
}
/* Конец демонстрационной программы */
```

Фирма ИНСАЙТ INSIGHT Comp.

Дистрибьютер **CLARION SOFTWARE** Corp.

CLARION TopSpeed

Версии 2.1 и 3.0 Pascal, Modula-2, C, C++

Идеальные средства создания заказного программного обеспечения.

Семинары CLARION User Group;
для официальных пользователей -
"горячая линия"; учебные курсы.

Предлагаем программное обеспечение фирм
Borland, Symantec, Microsoft, Corel, WordPerfect

103489, Москва, Зеленоград, корп. 602а
тел. 536-69-00 факс 534-93-46
E-Mail: puh@insi.msk.su | 2:5020/11

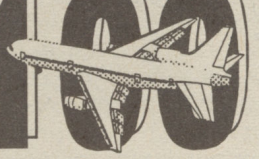
Дилер:
Харьков, СП "Энигма" т. (057-2) 21-74-24

INTERLINK

сети

Оборудование из США по Вашему заказу - за 15 дней

100 Mb/s



Беспроводные сети

COAX FDDI
TWISTED-PAIR ARCNET
FIBER-OPTIC ETHERNET

Streamer, UPS, Scanner, Plotter, Digitizer, Multimedia.
Продаем и устанавливаем сетевое оборудование.
Поставляем программное обеспечение NOVELL.
Разрабатываем топологию локальной сети.
Возможна установка техники у заказчика в пределах России.

205-6410, 205-6411, 205-6920,
205-6988, fax (095) 956-7021.
Москва, Глубокий пер. д. 5/11, под. 11

SIREX 486 ISA/EISA

Relcom® electronic mail

Крупнейшая информационная сеть СНГ даст Вам возможность общения с любым абонентом всемирных сетей Bitnet, MCI-Mail, Compuserv, Internet, EUnet и многих других.

Ваше сообщение, программа или графический файл дойдет до любой точки мира меньше чем за 3часа! В отличие от факса, не будет потеряно ни одной буквы.

Электронная почта— одна из самых современных технологий общения. Теперь она доступна каждому.

Используя возможности сети RELCOM, Вы можете обсудить проблему со всеми заинтересованными в ее решении людьми по всей планете одновременно, Вы можете извлечь информацию из сотен банков данных, получить свежую версию программы, даже рекламировать свою продукцию и услуги.

В сети RELCOM активно работают многие биржи, банки, крупные и небольшие предприятия, институты— всего более 60,000 на территории СНГ и далеко за миллион во всем мире. У абонентов RELCOM появляется возможность прямой связи с большинством крупных компьютерных, программистских и электронных фирм.

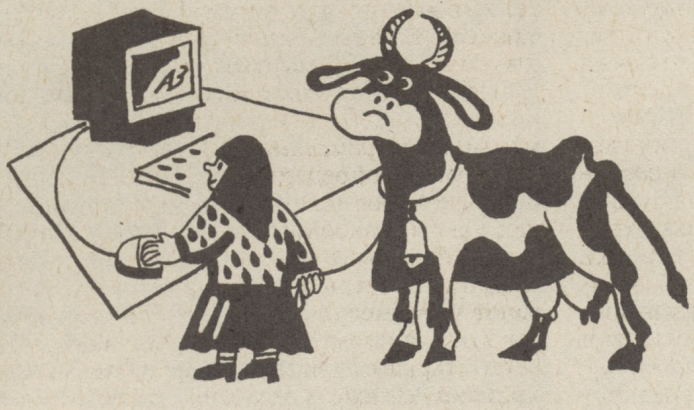
Единственный недостаток RELCOM — однажды начав работать в ней, Вы будете делать это всю жизнь.

demos™

Телефоны: (095) 231-21-29, 231-63-95, 233-05-92, 233-02-42
Факс: (095) 233-50-16
E-mail: info@hq.demos.su

Relcom является зарегистрированной торговой маркой фирмы Демос

Языки программирования четвертого поколения



А. Штрик

Часть 2. Характеристика некоторых распространенных языков 4GL

Приводимые ниже описания зарубежных (за исключением системы CASE-RT) языков и систем 4GL, подготовленные по материалам [1]-[4], демонстрируют существующие различия в подходах к построению и в функциональных возможностях языков четвертого поколения.

1. **Язык CASE-RT** обеспечивает проектирование и разработку многофункциональных программных прикладных систем, относящихся к классу управляющих, таких как, например, системы обработки данных от измерительных приборов, системы моделирования процессов, гипертекстовые справочные системы, обучающие системы и так далее. Окружение языка CASE-RT включает в себя технологический комплекс, предназначенный для интерактивного проектирования элементов прикладной системы и пользовательского интерфейса, а также библиотеки программ на языке С, которые входят в состав пользовательского задания. При этом поддерживаются функции прототипирования программной логики и интерфейса, моделирования и другие. Библиотеки программ содержат:

- функции управления, включая центральный диспетчер пользовательской системы, управление задачами, управление с помощью меню, а также «мышью» и графическими маркерами;
- функции, осуществляющие операции над спроектированными в технологическом режиме окнами и другими элементами интерфейсов; поддерживаются операции отображения, стирания, запоминания, восстановления изменения параметров окон;
- функции ввода-вывода: редактирование, отображение графиков и диаграмм, масштабирование и другие.

Данные функции используют программные элементы, спроектированные в технологическом режиме. При этом сначала в интерактивном режиме проектируются экранные формы: окна, тексты в окнах, структуры данных и виды их отображения. Далее проектируется схема управления системой в качестве таблицы «состояния-переходы». Система разбивается на множество состояний, каждому из которых ставится в соответствие исполняемая программа, множество возможных событий и множество реакций на эти события, то есть переходов в другие состояния. Каждому переходу ставится в соответствие последовательность действий, которые необходимо осуществить в процессе перехода. Среди них операции над окнами (формирование экрана) и над заявками на вызовы программ (организация вычислительного процесса). В процессе проектирования программы идентифицируются номерами и краткими описаниями.

Результат проектирования может рассматриваться как прототип программного приложения. Система CASE-RT поддерживает функции просмотра прототипа, при котором функционируют диспетчер, программа меню, программы формирования экранов, то есть все программы, кроме функциональных. Для гипертекстовых справочных систем и обучающих программ прототип фактически совпадает с самой системой, необходимо только собрать по определенным правилам загрузочный модуль. Для остальных прикладных систем после написания функциональных программ следует связать их адреса с указанными в проекте номерами по заданным правилам.

Аппаратная платформа функционирования — IBM PC с MS-DOS. Система разработана в РосНИИ ИТ и АП (г. Москва) в 1992 г.

2. **Система Accell/SQL** является системой 4 поколения для разработки прикладных программ, использующих реляционные СУБД Unify 2000, Oracle, Sybase, Ingres, Informix. Она

включает визуальный прикладной экранный генератор, управляемый прерываниями прикладной язык 4 поколения, механизм формирования отчетов, машину баз данных; при этом система поддерживает архитектуру «клиент/сервер». Язык системы Accell/SQL является англоподобным языком с непроцедурными операторами и с встроенными процедурными операторами; разрабатываемые программы собираются в прикладную программу и с помощью компоновщика связываются в единую машинную программу. Сюда же могут быть включены программы на языках C и COBOL. Основным достоинством системы Accell/SQL является непроцедурный программируемый интерфейс. Вместо того чтобы, как в других системах, редактировать, компилировать и тестировать разрабатываемые программы, здесь пользователь с помощью специального генератора в интерактивном режиме создает особые экраны. В библиотеке Accell/SQL содержится более 150 программных заготовок, которые предназначены для облегчения разработчикам процедуры заполнения таких экранов и программирования своих задач. Псевдогенератор инкрементно компилирует изменения, так что они могут быть протестированы без повторного проектирования.

Система Accell/SQL обеспечивает работу в интерфейсах OSF/Motif и OPEN LOOK. Кроме своей основной ОС UNIX, система Accell/SQL поддерживает также пять наиболее известных операционных окружений: Apple Macintosh, MS-DOS, MVS/CICS, VMS, Windows 3.0. При этом система обеспечивает интеграцию баз данных больших и персональных ЭВМ.

Основные области применения — это высокомобильные деловые приложения в правительственных, телекоммуникационных и финансовых системах, в промышленности и других областях. Система Accell/SQL используется во многих окружениях — их число достигает 200 аппаратных платформ, которые поддерживают символьно-ориентированные терминалы, Motif, OPEN LOOK, Windows 3.0. Любое приложение, написанное с помощью Accell/SQL, может быть перенесено на другую платформу без каких-либо изменений.

Основная системно-аппаратная платформа — UNIX; фирма-разработчик — Unify Corp.; первая установка относится к 1986 г., в настоящее время зарегистрировано свыше 8,5 тысяч пользователей.

3. Система **FOCUS** интегрирует язык 4 поколения и СУБД, поддерживая как реляционные, так и иерархические структуры данных; при этом обеспечиваются такие возможности, как мультисписочность, параллельный доступ, переносимость между разными аппаратными платформами, универсальный доступ к данным. Система ориентирована на использование в широкой области приложений, включая интерактивную обработку транзакций, системы поддержки решений, деловые и финансовые системы.

Язык FOCUS 4GL может быть использован, чтобы реализовать обработку любых структур запросов к данным и формирование отчетов. Язык также обеспечивает прозрачные и переносимые программы для различных платформ, предоставляя возможность как распределенной, так и параллельной обработки данных. Предусмотрен автоматический генератор кода для получения прикладных программ, схем экранов и систем представления данных. Основные достоинства системы FOCUS связаны с переносимостью прикладных программ на широкий спектр аппаратных платформ, с возможностью обработки файлов данных из разных типов баз данных, с поддержкой широкой области прикладных приложений, в том числе и архитектуры «клиент/сервер» с распределенной обработкой. В числе известных недостатков системы — необходимость создания интерфейса конечного пользователя, особенно для начинающих программистов, а также наличие ограничений на размеры имен файлов и данных, на объемы памяти и число файлов.

FOCUS/4GL является англоязычным непроцедурным языком; обеспечена поддержка SQL путем автоматической трансляции SQL каталогов в эквивалентные файлы-определители FOCUS. Предусмотрено интерактивное текстовое редактирование, включающее как экранное заполнение, так и оконное, причем без необходимости кодирования. Имеются средства создания отчетов и средства анализа, которые окружают непроцедурный язык с опциями для форматирования, используемыми по умолчанию. Предусмотрена графика с высоким разрешением, включающая индивидуальные указатели, гистограммы, штриховые диаграммы; имеются средства статистического анализа для таких функций, как временной анализ, регрессионный анализ, факторный/дискриминантный/вариантный анализ. В состав системы входит Lotus подобная электронная таблица, а также хранилище данных и ряд других средств.

Обеспечивается работа со структурами всех основных баз данных (ADABAS, CA-DATACOM/DB, CA/IDBS, DB2, IMS, dBASE III, CISAM, ULTRIX/SQL, Unify). В числе поддерживаемых языков программирования — C, COBOL, Fortran, Pascal, Ada, Assembler.

Аппаратная платформа: большие ЭВМ IBM, IBM AS/400, IBM PC, VAX, Hewlett-Packard, Wang VS, а также большое количество UNIX ориентированных миникомпьютеров и рабочих станций. Фирма-разработчик — Information Builder, Inc.; установлено свыше 7000 систем на больших ЭВМ и свыше 150 тысяч на персональных ЭВМ.

4. **GENER/OL** является системой 4GL для создания приложений с интерактивной обработкой запросов для платформы IBM. GENER/OL позволяет избежать цикла долгой разработки прикладной программы, и, хотя система GENER/OL обычно функционирует под управлением системной среды CICS, от пользователя не требуется знания командного языка CICS. Система GENER/OL использует англоподобный высокоуровневый язык программирования и форматированный экран.

В состав системы входят следующие компоненты:
- высокоуровневый интерактивный процедурный язык четвертого поколения;
- редактор текстов и компилятор с этого языка;
- разделяемые компоненты системы (редактор маски, определители данных, экран дисплея);
- утилиты для поддержки и сопровождения прикладных программ;
- средства для формирования отчетов;
- средства защиты данных;
- средства связи прикладных программ GENER/OL с прикладными программами, написанными на других языках.

Работа с системой GENER/OL основана на применении меню, когда разработчик работает с системой, используя форматированный экран. Работая с этими экранами, даже новичок может строить дисплейные карты, редактировать маски, управлять областями памяти. Предусмотрены интеллектуальные средства обеспечения немедленного обнаружения программной ошибки; имеется специальный входной редактор, контролирующий входные данные; обеспечена полная поддержка инкрементной компиляции, при которой только изменение или корректировка строк кода требуют перекомпиляции.

Есть возможность включения в состав проектируемой программы компонентов, написанных на языках COBOL, RPG, PL/1 и ряде других. Язык GENER/OL позволяет пользователю вызывать графические подпрограммы IBM (GDDM, PGE, ICU) как динамически, так и интерактивно без их компиляции или компоновки. Хотя в системе GENER/OL отсутствует традиционный для CASE систем репозиторий, необходимая информация содержится в специальных библиотеках, которые позволяют пользователям постоянно хранить и накапливать в них создаваемые компоненты и при необходимости с помощью утилит Системного Администратора формировать необходимые их конфигурации.

Прикладная область исключительно широка: от самых узких одноцелевых приложений (например, задачи расчета амортизации средств) до больших ответственно-критических операций типа управления производственными процессами. Наиболее удобна система GENER/OL для приложений, где требуется доступ к удаленным данным или работа с большим числом поколений баз данных, например, обработка банковских и финансовых запросов, процессы мониторинга, контроль инвестиций и так далее. При этом система не требует использования специальной СУБД, а ориентирована на широко известные: VSAM, DB2, SQL/DS, IMS/DL1, ADABAS, DATACOM, TOTAL.

Аппаратная платформа: IBM System 370, 390, 30XX, 4300, 9370 под управлением DOS/VSE, VSE/SP, VSE/ESA, MVS/370, MVS/XA, MVS/ESA.

Фирма-разработчик — Pansophic Systems, Inc.; первая установка относится к 1982 г., в настоящее время около 1000 пользователей.

5. Empress 4GL — большая система управления реляционными базами данных, занимающая около 8 Мб дискового пространства. Основные функциональные возможности системы: экранное раскрашивание, перекрывающиеся окна, определение функциональных клавиш, возможность управления вычислениями с помощью прерываний. В числе дополнительных технологических средств: поддержка ядра OSF/Motif в X-Windows, поддержка экрана с двоичной графикой, интеграция изображений/звуков, поддержка «мыши». Работа с содержимым базы данных осуществляется с помощью специальных редакторов, позволяющих вводить данные в базу, модифицировать и удалять их. Система предоставляет пользователю средства работы с окнами и формами больших размеров. Для создания меню в системе Empress нужно определить окно, в котором будет появляться это меню, форму меню и поля ввода-вывода данных, которые могут быть запрограммированы средствами языка, а также сценарий, задающий условия появления меню и реакцию на сигнал от клавиатуры. Можно запрограммировать функциональную клавиатуру (изменение размеров окна, удаление записей из базы данных, вызов команд операционной системы и другое).

Важным компонентом системы является генератор отчетов, который позволяет создавать отчеты на основе данных из произвольного числа таблиц базы данных или из любых текстовых файлов. Система Empress 4GL поддерживает стандартный SQL, имеет средства работы с сетями, а также хорошие возможности по обработке звуковых данных и по графике. Ядром системы является язык четвертого поколения M-Builder, который отличается особой гибкостью, позволяя пользователю запускать программы в любой момент работы системы; при этом инициализацией может служить не только нажатие клавиши, но и положение курсора на экране. Язык позволяет работать с окнами (перемещать, изменять размеры) и с базой данных, а также производить обмен между базой данных и экраном. Из системы можно обращаться к программам, написанным на других языках, в частности, на C и Fortran.

Аппаратная платформа — IBM PC/XT/AT, IBM PS/2, Compag Deskpro 286/386, DEC microVAX; операционные системы — Windows 3.0, MS-DOS, UNIX, VMS. Разработчик — фирма Empress Software,Inc.; первая установка относится к 1981 году, в настоящее время количество зарегистрированных пользователей около 20 тысяч.

6. Система Ingres/4GL в связке с СУБД Ingres является полнофункциональным окружением языка четвертого поколения, предназначенного для разработки переносимых, управляемых с помощью меню многопользовательских приложений для самых разных прикладных сфер. Средства языка решают проблемы информационного управления разработкой приложения, резко сокращая затраты на прототипирование, создание и использование новых прикладных программ. Система Ingres первоначально проектировалась для использования на платформе VAX/VMS фирмы DEC. В настоящее время охватывает широкий спектр машин: Hewlett-Packard (MPE/XL), IBM PC, Apple Macintosh, а также более чем 35 UNIX/Ultrix систем. Построенная на стандарте ANSI SQL, система Ingres включает в свой состав язык 4GL и объектно-ориентированный интерфейс пользователя. Кроме того, в Ingres входят средства обработки изображений, средства генерации программ с языка четвертого поколения, англоподобный язык запросов, развитые возможности записи/считывания данных; она обеспечивает операции связывания и компоновки программ, имеет удобные средства как для начинающих пользователей, так и для опытных программистов, а также интерактивную обработку транзакций и возможность создания распределенных баз данных. Используя эти компоненты, можно создавать переносимые программы, что достигается применением таких экранных операций, как атрибуты дисплея, области просмотра и функциональные ключи без какого-либо их программирования. Благодаря этим средствам создаваемые программы оказываются переносимыми более чем на 100 типов дисплеев.

Поддержка распределенной обработки дает возможность использовать систему в гетерогенной конфигурации баз данных, обеспечивая доступ данных от

любой удаленной точки обработки. Ingres является одной из первых систем с мультисерверным управлением данными, которое опирается на параллельную обработку данных как в мультипроцессорных системах, так и в локальных сетях. В числе сильных сторон пользователи системы отмечают возможность работы на многих популярных аппаратных платформах; наличие графического интерфейса пользователя (GUI), поддерживающего многооконную разработку; развитые средства распределенной обработки. В то же время отсутствует поддержка интерфейса с такой популярной базой данных, как DB2 фирмы IBM, что ограничивает возможности использования Ingres в приложениях, где установлены системы IBM и VAX.

Среди поддерживаемых языков программирования — все современные языки: C, Cobol, Fortran, Basic, Pascal, PL/1, Ada; среди интерфейсов такие распространенные, как DECwindows, OSF/Motif, Open Windows, Windows/386, Presentation Manager, Windows 3.0, Macintosh.

Аппаратная платформа: DEC VAX, Apollo Domain, AT&T 3B System, Data General MV Family, IBM System/370, NCR Tower 32, Unisys U Series и ряд других ЭВМ. Разработчик — фирма ASK Computer Systems, Inc.; первая установка в 1981 г., в настоящее время около 7,5 тысяч пользователей.

7. **Natural** — система поддержки языка четвертого поколения, обладающая большими возможностями как для разработки прикладных приложений, так и для применения конечными пользователями. В числе этих возможностей: интеллектуальный язык; проектирование экрана; редакторы отчетов и глобальной области данных; структурированный синтаксис; автоматический интегрированный словарь данных; интерактивный компилятор; система сопровождения библиотек, а также гибкий и отработанный язык программирования четвертого поколения. Комбинация возможностей разработки и функций языка 4GL предоставляет прикладному разработчику все необходимые средства для создания промышленных приложений с дружественным интерфейсом, который доступен и для программ на языках третьего поколения. В числе дополнительных технологических функций — поддержка оконной техники и описание экрана. Кроме того, окружение Natural предусматривает поддержку меню-ориентированных отчетов, создание и сопровождение пользовательских баз данных, графическое представление баз данных и ряд других возможностей. Обеспечивается работа со следующими базами данных: ADABAS, DB2, SQL/DS, IMS/DB, DL/1, а также файлами в формате VSAM.

Известно целое семейство в системе Natural, которое отличается аппаратно-системной поддержкой, а также выполняемыми функциями: Natural, Natural Construct, Natural DB2, Natural ISDF, Natural Operations, Natural OS/2.

Так, Natural Construct — это прикладная система генерации деловых приложений. Обеспечена возможность быстрого прототипирования и разработки полного приложения. Все функции выполняются в интерактивном режиме, включая разработку спецификации, генерацию кодов и тестирование. Автоматически генерируются приложения, работающие с ADABAS, DB2, SQL/DS или с файлами в формате VSAM. Аппаратная платформа — IBM 370 и DEC VAX.

Natural DB2 — это интегрированное окружение для разработки прикладных приложений для базы данных DB2, которое объединяет функциональные требования прикладной системы и возможности языка Natural, чтобы ускорить разработку приложения для работы в DB2. Аппаратная платформа — IBM 370. Natural OS/2 — современная система, обеспечивающая возможность разработки приложений для персональных ЭВМ с операционной системой OS/2.

Разработчик — фирма Software AG of North America, Inc.; первая установка осуществлена в 1979 году, последняя версия для работы в OS/2 выпущена в 1990 году.

8. Язык **NOMAD** является одним из наиболее популярных языков четвертого поколения и обладает широким спектром функциональных возможностей. Создатели языка — отделение транснациональной электронной и аэрокосмической корпорации Thompson SA с центром во Франции. В настоящее время появилась новая версия этого пакета 5.0, которая имеет целый ряд новых возможностей, в частности, такие как сохранение временных данных в оперативной памяти, переопределение принятых по умолчанию функциональных клавиш, возможность перемещения фрагментов данных между окнами.

В составе NOMAD имеется специальная программа, управляющая ресурсами процедур и команд и информирующая пользователя об использовании памяти. Система может работать с внешними базами данных. Важной функцией является словарь данных, обеспечивающий экранное порождение и поддержку схемы базы данных; автоматическая генерация словаря и отчета по описанию схемы базы данных; построение файла перекрестных ссылок базы данных и приложений. Система также обеспечивает несколько методов управления различными уровнями доступа к базе данных. На уровне схемы могут быть установлены, в частности, параметры, разрешающие любые изменения данных, запрещающие изменения существующих значений, запрещающие добавление новых значений, запрещающие удаление существующих данных.

Для доступа к данным предоставляется ряд команд, например, команда LIST может извлекать данные из файлов SQL/DS, DB2, IDMS, IMS и строить отчеты. LIST работает как с реляционными, так и с иерархическими базами данных; имеется возможность прототипирования отчетов по результатам работы этой команды. Для создания новых баз данных предусмотрена команда CREATE, которая перекачивает данные из исходной базы в соответствии со спецификацией. Она также создает скомпилированное определение схемы. Созданная база данных может обрабатываться всеми реляционными операциями и средствами выборки. Предусмотрен ряд команд для отладки программ в среде NOMAD.

Имеются развитые средства работы с отчетами, поддерживаются функции обработки информации для статистического и финансового анализа и принятия решений. Предусмотренный интерфейс программиста позволяет писать прикладные программы на языках COBOL, Fortran, PL/1 и на Assembler IBM 360/370. Для работы в среде ПЭВМ имеется специальная версия PC NOMAD, реализованная на языке C. Эта система поддерживает работу в оконном стиле, которая предоставляет средства каталогиза-

ции наиболее часто используемых баз данных, отчетов и любых других средств. Кроме того, могут быть получены отчеты с различными опциями сортировки и вычислений. В середине 90-х годов ожидается многопользовательская версия NOMAD для ПЭВМ, работающая с серверами SQL Base и SQL Server.

Аппаратная платформа функционирования: DEC VAX, IBM System/390, IBM PC, IBM PS/2. Язык и система разработаны фирмой Must Software Int'l в 1983 году, общее число пользователей оценивается в 350 тысяч.

9. **PRO-IV** — язык четвертого поколения, который поддерживает разработку, реализацию, исполнение и сопровождение прикладных задач в сфере бизнеса. Построение языка PRO-IV базируется на концепции, которая фундаментально отличается от тех, что заложены в другие языки программирования. Программы на PRO-IV не компилируются, подобно языкам третьего поколения, и не интерпретируются, как большинство других языков IV поколения. Основу работающей прикладной программы на PRO-IV составляет набор резидентных модулей, находящихся в оперативной памяти. Из этих модулей компонуется прикладная программа, которая управляется таблицей, автоматически генерируемой во время ввода информации в систему пользователем. Пользователь работает с системой PRO-IV через помогающие в разработке меню. Возможности системы PRO-IV обеспечивают:

- создание и модификацию описаний данных и файлов;
- создание и модификацию меню, экранов ввода и вывода, отчетов, прикладных функций;
- печать данных, файлов и документирование прикладных функций;
- создание и модификацию описаний правил защиты данных и ограничения доступа к различным утилитам.

Система PRO-IV позволяет разработчикам достичь значительного повышения производительности труда на стадиях разработки и сопровождения прикладных программ. При использовании PRO-IV разработчики избавлены от многих этапов, свойственных работе на традиционных языках программирования. В системе PRO-IV используются простые в применении, напоминающие бланки для ввода данных экраны, которые помогают работе пользователя на каждом этапе. Развитая система диалога, реализованная в структурной среде поддержки PRO-IV, превращает самую сложную работу в совокупность простых шагов. Язык обеспечивает возможности переноса приложений более чем на 70 различных платформ, включая все основные типы ЭВМ и операционных систем. При этом поддерживается программный интерфейс с CASE системой ProKit*WORKBENCH, обеспечивающий автоматический перенос проектной информации, разработанной в ProKit*WORKBENCH на начальных этапах жизненного цикла, в систему PRO-IV. Это позволяет поддерживать весь жизненный цикл информационной системы.

Аппаратная платформа: AT&T 3B Systems, DEC VAX, Bull DPS 8/88/8000, IBM AS/400,IBM System/3X, IBM System 370, NCR Tower 32. Разработана фирмой McDonnell Douglas Systems Integration Company в 1986 году; количество пользователей — свыше 2,5 тысяч.

10. **RAMIS** — развитая система для конечного пользователя, которая обеспечивает обработку запросов, подготовку отчетов и создание прикладных административных приложений. Она содержит интегрированную систему, включающую внутреннюю СУБД, дополнительные файлы и реляционные данные, «бесшовную» связь с IBM PC, возможности сопровождения данных, опции подготовки отчетов, включая графический и финансовый анализ, прототипирование и разработку приложений с получением скомпилированного кода. Известно две версии системы: RAMIS IS для IBM System/370 и RAMIS/PC для IBM PC, IBM PS/2.

RAMIS IS — это специализированная группа средств для конечного пользователя, чтобы решать проблему очередей, подготовки отчетов, а также создавать окружение разработки. Эти средства позволяют проектировать интегрированные системы, которые включают внутренние СУБД в состав приложений. Доступ ко всем возможностям в RAMIS IS происходит через командное меню, использующее командный синтаксис RAMIS. Подсказка пользователям идет через контекстно-чувствительную систему. С помощью средств, использующих технологию экспертных систем, имеется возможность разрабатывать полное меню системы, применяя соответствующие шаблоны. Полноэкранный редактор также с использованием методов искусственного интеллекта позволяет пользователям создавать и модифицировать файл дескрипторов RAMIS, обеспечивая таким образом доступ RAMIS к файлам и базам данных форматов VSAM, QSAM, IMS, IDMS/R, DB2, SQL/DS, TERADATA, DATACOM/DB, ADABAS.

RAMIS/PC является информационной управляющей системой, которая помогает пользователям персональных ЭВМ управлять информацией в своих системах. Все функции этой системы реализуются через пользовательское меню. Предусмотрена связь с большой ЭВМ. Это достигается выбором англоязычного или меню-ориентированного генератора отчетов. Предусмотрены средства для создания сложных PC-ориентированных приложений и прототипов этих приложений на больших ЭВМ.

Разработка фирмы On-Line Software International, Inc. Выпущена в 1987 году, количество зарегистрированных установок 1440.

11. Система **Sapiens** является современным средством разработки и эксплуатации прикладных программ, обеспечивающим возможность быстрого создания эффективных приложений в среде IBM/EC ЭВМ. Система разработана с использованием достижений в области баз знаний и объектно-ориентированного программирования; поддерживает стандарт SAA (Systems Application Architecture). В системе реализован декларативный подход, который, используя методы искусственного интеллекта, позволяет накапливать в виде базы знаний набор фактов о проектируемой программе и принципах ее разработки, чем существенно снижается число возможных ошибок. Система обеспечивает многократное повышение производительности труда при разработке и сопровождении прикладных систем за счет широкого использования баз знаний, генераторов программ, моделирования, автоматического документирования и тестирования.

При использовании системы Sapiens прикладные программы рассматриваются как набор фактов и правил, которые описывают последовательность обработки информации. Разработка прикладной программы представляет собой непрерывный процесс записи этих фактов и правил в базу знаний. Эта база содержит всю необходимую техническую информацию для перевода поступающих фактов и правил в прикладную программу для ЕС ЭВМ. База знаний интегрирует факты и правила, относящиеся к данной прикладной области. Она может наполняться в диалоговом или пакетном режимах с помощью специальной программы моделирования или через какие-либо другие инструментальные системы, работающие со стандартным Хранилищем IBM.

Объективное измерение возможностей Sapiens было проведено независимыми экспертами при оценке большой группы языков четвертого поколения. Во время однодневного испытания, где было представлено около 30 генераторов прикладных программ, требовалось написать программу, создание которой на Cobol занимает около 14 часов. Системе Sapiens потребовалось всего 3 часа, тогда как большинство других испытываемых систем не успело завершить работу за полный день испытаний. Система Sapiens широко используется в банках, страховых компаниях, в промышленности, на транспорте, в торговле, в государственных и промышленных предприятиях.

Системная платформа — любые современные версии ОС ЕС ЭВМ. Фирма-разработчик — Sapiens International (Англия).

12. Система **Thoroughbred IDOL-IV** является интегрированным окружением разработки для создания многопользовательских приложений в области бизнеса, причем эти приложения могут полностью создаваться на языке четвертого поколения без использования традиционных языков третьего поколения C, Cobol, Pascal. Процедура установки системы проста, но требует создания специальной директории /usr/lib/basic. Максимальный размер записи 32 Кб, а их количество на базу данных 8 млн.

Система Thoroughbred IDOL-IV обладает широкими функциональными возможностями. Включает централизованный словарь данных и систему управления базой данных с генератором отчетов и окружением языка четвертого поколения. Имеет специализированный генератор отчетов, поставляемый отдельно и допускающий отчеты с данными из нескольких файлов. При этом формат отчета аналогичен форматам отчетов в языке Cobol. Язык системы основан на BASIC и отличается возможностью доступа из программы ко всему содержимому базы данных — форматам, словарю данных, сообщениям. Из программ этой системы возможно обращение к программам на BASIC. Сценарии системы Thoroughbred IDOL-IV могут быть вызваны из соответствующих программ. Недостатками системы являются ограничения на размеры программы (64 Кб для UNIX и 54 Кб для DOS), на размещение описаний данных и заголовков и другие.

Аппаратно-системная платформа функционирования — IBM PC, IBM PS/2; MS-DOS, UNIX, OS/2. Фирма-разработчик — Concept Omega Corp. Выпущена в 1988году и имеет около 800 пользователей.

13. Язык **UNIFACE** отличается тем, что в нем реализован принципиально новый подход к проектированию системы обработки документов. Он основан на прямой связи между определениями в словаре данных и способом построения форм. Описания форм отражают определения сущностей и связей между ними. Основные компоненты построения форм в UNIFACE называются фреймами, которые подобны окнам и могут быть определены для поля, сущности и целой формы. Обработка формы определяется взаиморасположением и вложенностью форм друг в друга. Перед тем как создавать формы, все поля, таблицы и другие сущности должны быть описаны в словаре, который соответствует стандарту ANSI. Поэтому все описание, называемое схемой, имеет три уровня: внутренний (определение таблиц, полей, индексов на физическом уровне), концептуальный (логические определения полей и сущностей) и внешний, соответствующий формам.

UNIFACE поддерживает несколько типов реляционных СУБД, предоставляя пользователю возможность одновременной работы с данными из разных СУБД и гарантируя целостность всех данных. (Например, таблица «Поставщик» может быть реализована на СУБД ORACLE, а таблица «Товар» на СУБД Rdb). Целостность данных определяется на уровне словаря данных, который гарантирует, что при удалении поставщика из таблицы удаляются соответствующие товары. Это тем более важная особенность, что, как правило, в крупных организациях функционируют системы, построенные на разных СУБД и при необходимости их объединение в систему UNIFACE может оказаться эффективным.

Формы в UNIFACE строятся с помощью специального средства и состоят из фреймов, связанных с полями и сущностями. Процесс создания контролируется системой на основе словаря, доступного в любой момент. В среднем разработка одной формы занимает около часа. Основным назначением UNIFACE является разработка приложений, основанных на формах. Средства генерации отчетов позволяют создавать относительно простые отчеты, а для более сложных можно использовать другие системы. Одна из версий системы UNIFACE входит в состав реляционной СУБД Sybase под названием Fastbuild. Благодаря своей способности работать с несколькими СУБД система UNIFACE может использоваться для построения открытых систем.

Аппаратная платформа функционирования — IBM PC и совместимые модели с памятью не менее 2 Мб, а также среда VAX. ■

Литература

1. How Europe is using CASE// Datamation.– August 1, 1990.
2. Н.П. Викторова. Система поддержки разработки интерактивных и управляющих программ./ Семинар «CASE-технология»: Сб. материалов, Москва, 1992.
3. Языки четвертого поколения// Программные продукты и системы.– 1992.– Вып. 1.
4. Forth-Generation Language Software Product Profiles// DATAPRO.– October 1991.

На выставку в северную столицу!

Как правило, упоминание о выставке компьютерных технологий прочно ассоциируется с московским ВВЦ (бывшая ВДНХ) или с выставочным комплексом на Красной Пресне. Да и пресса пишет в основном о выставках Комтэк или Софтул (и наш журнал не исключение). Но вот в конце 1992 года наши представители побывали на выставке в Санкт-Петербурге и там познакомились с рядом фирм, в том числе и московских, которые очень высоко отзывались как об организации «ИНВЕКОМ-92», так и об ее организаторах.

Надо признать, что в Петербурге нам понравилось, и с этого времени завязались контакты журнала «Монитор» с АО «Рестэк». Наша редакция прилагает все усилия, чтобы не пропускать мероприятия, проводимые фирмой. Акционерное общество «Рестэк» еще с 1991 года специализируется на организации выставок и ежегодно проводит в Санкт-Петербурге на базе выставочного комплекса «Ленэкспо» две крупномасштабные компьютерные выставки. Одна из них обычно проходит летом и представляет собой специализированную экспозицию в области АСУ и средств обеспечения автоматизации. Вторая выставка — «Информатика и вычислительная техника» («ИНВЕКОМ») — традиционно проводится в декабре.

Тематика выставки «Информационные автоматизированные системы», проходившей в выставочном комплексе в Гавани с 11 по 15 мая 1993 года, определялась общей Программой информатизации России. Задачей выставки являлась помощь городским службам, банкам, биржам, предприятиям в автоматизации их деятельности. Посетители выставки могли познакомиться с программным обеспечением для систем планирования, управления, автоматизации исследований и управления производственными процессами, системами контроля качества и системами передачи данных. По срокам выставка совпала со второй Международной Санкт-Петербургской конференцией «Региональная информатика-93». По оценкам организаторов, выставка прошла более чем успешно. Достаточно сказать, что если в прошлом году на выставке было 64 участника, то уже в этом году в Санкт-Петербург съехались 108 фирм из России, стран СНГ, ближнего зарубежья. Пришлось (но это, конечно, приятные хлопоты) увеличить и выставочные площади — в этом году выставка занимала уже два павильона.

Не меньшей популярностью пользуется и выставка «ИНВЕКОМ». Так, в 1992 году в ней приняли участие более 150 фирм. Среди ставших уже привычными разделов появился новый — «Обучающие технологии и системы», который вызвал живой интерес как у участников, так и у посетителей. По мнению АО «Рестэк», главное достоинство выставки «ИНВЕКОМ» заключается в том, что на ней широко и достаточно равномерно представлена такая разнообразная тематика, как программные средства и системы, компьютеры и сети, техника связи и системы телекоммуникации, техника и оборудование для офиса, обучающие системы.

С точки зрения редакции журнала, исключительно важным является присуждение почетных дипломов лучшим экспонатам выставки. И дело ведь совсем не в том, чтобы польстить участнику и уговорить его приехать снова. Ведь отечественные разработки вынуждены (извините, но рынок есть рынок!) конкурировать с западными гигантами компьютерной индустрии: IBM, Borland, Novell и так далее. И чтобы достойно выглядеть в одном ряду с такими соперниками, надо работать! Другого рецепта нет. Для нас с Вами это означает, что отечественные фирмы вынуждены неуклонно повышать качество своей продукции, что, согласитесь, можно только приветствовать.

Популярность проводимых АО «Рестэк» выставок из года в год продолжает расти. С момента первой выставки (1991 год) более чем в три раза увеличилось не только число участников и выставочной площади, но и количество заключаемых на выставке контрактов. Большинство фирм, показывавших свои экспонаты на одной из выставок, становятся ее постоянными участниками.

Читателям журнала, должно быть, интересно узнать, что фирма «Рестэк», выросшая из регионального инженерного центра «Ленинформатика», прилагает значительные усилия для поддержки научных организаций, занимающихся перспективными разработками, а также организует проведение различных конференций и семинаров. «Рестэк» помогает и тем молодым коллективам, которые пока недостаточно крепко стоят на ногах, но предпочитают решение важных научных и технических проблем торговле «Сникерсами». Таким фирмам «Рестэк» оказывает спонсорскую поддержку, обеспечивая их участие в специализированных выставках, чтобы они имели возможность установить деловые контакты, продвинуть свои товары и услуги на рынок.

Вам еще не поздно заявить о своем участии в мероприятиях, проводимых АО «Рестэк». Например, можете принять участие в выставке «Региональная информатика», которая пройдет в рамках международной конференции «РИ-94» в мае 1994 года.

АО «Рестэк» приглашает читателей «Монитора» посетить выставку «ИНВЕКОМ-93», проходящую в Санкт-Петербурге с 7 по 11 декабря 1993 года.

Телефоны: (0812) 112-29-48, (0812) 112-23-48. ■

СОЛИДНЫЙ ПОСТАВЩИК

Мы существенно обновили и расширили ассортимент. У нас появилось несколько новых партнеров среди крупнейших производителей компьютерной техники.

Мы готовы предложить весьма привлекательные цены на продукцию всемирно признанных марок.

Мы надеемся, что вслед за тысячами наших клиентов и Вы сможете убедиться в нашей репутации стабильного поставщика наиболее передовых компьютерных систем.

ЛАЗЕРНЫЕ И СТРУЙНЫЕ ПРИНТЕРЫ

HEWLETT PACKARD

LaserJet 4L ★ 4 ★ 4P ★ 4Si
Новейшее семейство лазерных принтеров.
DeskJet 510 ★ 500C ★ 550C
Струйные принтеры, 300dpi.
DeskJet 1200C ★ 1200C/PS
Цветные струйные принтеры, 300 dpi (черный текст до 600dpi).

А также любое оборудование из текущего ассортимента компании Hewlett-Packard.

ГРАФИЧЕСКИЕ СТАНЦИИ ДЛЯ КОМПЬЮТЕРНОЙ АНИМАЦИИ

TRUEVISION

Truevision Targa+16,32 PAL+free Tips for Targa
Графический адаптер, RGB видеовыход, палитра 16 млн. цветов, редактор Tips.
Truevision ATVista 4 Mb 60 MHz+free Tips for ATVista
Видеографический адаптер, встроенный графический процессор, редактор Tips.
VideoVGA 16 Bbp PAL
Скан-конвертор SVGA to PAL, S-Video.
Autodesk 3D Studio v.2.01a + free Ipas Toolkit
Система пространственного моделирования и анимации.
Средство программирования IPAS процессов.
Autodesk IPAS Processes for 3D Studio
Пакет спецэффектов IPAS для Autodesk 3D Studio.
Fractal Design Painter (MS Windows) v.2.0
Пакет для двухмерного рисования в стиле natural media.

Пожалуйста, обращайтесь за описанием полного графического комплекса на базе IBM PC.

МОНОХРОМНЫЕ И ЦВЕТНЫЕ СКАНЕРЫ

DEXTRA

DextraReader 300dpi ★ 600dpi
Монохромные сканеры, формат А4.
DextraPage DF-1200 ★ DF-2400
Цветные планшетные сканеры, 24bit, 1200dpi ★ 2400dpi, формат А4.

Мы являемся эксклюзивным представителем компании Dextra Corporation в странах СНГ.

КОПИРОВАЛЬНЫЕ АППАРАТЫ

XEROX

XEROX 5220
XEROX 5317/1 ★ 5317/2
XEROX 5331/1 ★ 5331/2 ★ 5331/3
XEROX 5340 ★ 5350

А также любое оборудование из текущего ассортимента компании Xerox.

КОМПЬЮТЕРЫ КЛАССА NOTEBOOK

orange

NOTEBOOK 486SX-25/4/120/Mono
NOTEBOOK 486SX-25/4/120/Color Dual Scan
NOTEBOOK 486DX-33/4/120/Color TFT active matrix
NOTEBOOK 486DX-66/4/120/Color Dual Scan
NOTEBOOK 486DX2-66/4/120/Mono LCD

STEEPLER

Л У Ч Ш Е В М Е С Т Е

VISA MasterCard AMERICAN EXPRESS

Россия, Москва, Пречистенка, 40

246-81-92 **245-63-36** 246-24-99

Современные модемы для современных телекоммуникаций

Не нужно доказывать, что информатизация требует развития телекоммуникаций.
Не нужно доказывать, что современные телекоммуникации требуют использования качественных модемов.
Мы много лет занимаемся развитием телекоммуникаций в нашей стране, и сейчас предлагаем только лучшее. Суперсовременные модемы ZyXEL, Telebit, US/Robotics, универсальные модемы Discovery, выдержавшие проверку временем, любые компьютеры для телекоммуникационных систем.

Discovery 2400CM/D — внешний модем для надежных телекоммуникационных систем общего назначения. Аппаратная реализация MNP5 и V.42bis. Отправка факсов. Сертификат Минсвязи. Гарантия 2 года.

Discovery 2496HX — встраиваемый факс-модем, отлично подходящий для российского офиса. Прием и передача факсов, аппаратная реализация MNP5 и V.42bis.

Discovery 2496PX — карманный факс-модем для тех, кто работает всегда и везде. MNP5, V.42bis, питание от батареи 9В или от сети.

Telebit Worldblazer — скоростной факс-модем (до 70,000 бит/с), прославившийся своей надежностью, в том числе при работе на плохих линиях.

ZyXEL U-1496 — высокоскоростные модемы (до 57,600бит/с) с возможностью передачи и приема факсов и речевых сообщений в цифровой форме.

demos™

Телефоны: (095) 231-21-29, 231-63-95, 233-05-92, 233-02-42, 231-60-02
Факс: (095) 233-50-16
E-mail: info@hq.demos.su

Знак и логотип **demos** — торговая марка фирмы Демос

Pascal сегодня, вчера, завтра (Аналитический обзор)

М. Трофимов

Настоящий обзор посвящен месту, занимаемому языком Pascal среди других языков программирования. Основой для этого послужили свежие зарубежные источники, в том числе и нетрадиционные — компьютерные сети: internet, bitnet, earn и suearn. Используется также и русскоязычная литература. Обзор адресован в первую очередь программистам-практикам, работающим на IBM PC под управлением MS-DOS, поэтому мы будем в основном придерживаться этой компьютерной платформы, в особых случаях отмечая совместимость того или иного продукта с другими платформами: время течет быстро, и ассортимент машин, доступных читателям «Монитора», растет, поэтому вопрос о переносимости своих программ, пусть и в будущем, с компьютера одного типа на другой начинает волновать многих. Обзор предваряет сравнительный анализ различных компиляторов языка Pascal под управлением MS-DOS, который планируется сделать в лаборатории тестирования Monitor TestLab в самое ближайшее время.

Немного истории

Начнем со вчера. На второй конференции «Истории языков программирования (HOPL-II)», проходившей в Кэмбридже (Массачусетс, США) 20-23 апреля 1993 г. под эгидой Группы языков программирования Американской компьютерной ассоциации (ACM SIGPLAN — Special Interest Group of Programming Languages of the Association for Computing Machinery), создатель языка Pascal Никлаус Вирт начал свой доклад следующими словами [1].

«Язык программирования Pascal был разработан в 1968-1969 годах, и я назвал его именем французского философа и математика, который в 1642 году создал приспособление, явившееся одним из первых цифровых арифмометров. Первый компилятор языка был сделан в начале 1970 г., тогда же было опубликовано описание языка [2]».

Итак, история языка Pascal начинается с указанных дат, однако вслед за Виртом стоит кратко остановиться на предыстории. В докладе Вирт отмечает два основных «допаскалевских» направления — это Fortran, широко распространенный в то время, «но пренебрегающий систематичностью структуры» [1], и Algol-60, уделивший больше внимания систематичности программирования и структуре программы. Кроме того, по образному выражению Вирта, в 60-х можно было наблюдать «каньон между научным и коммерческим программированием». Одним берегом этого каньона был Fortran, другим — Cobol. Cobol также послужил источником важных идей для дальнейшего развития языков программирования, в частности, это структуры-записи (records).

Первый серьезный шаг к Pascal был сделан в 1966 г., когда в Стэндфордском университете был создан Algol-W, где воплотились идеи Хоара по связыванию ссылок и виртовская концепция динамических структур данных. Хотя «по сути, — говорит Вирт, — Algol-W был расширением Algol-60» [1].

Общеизвестно, что Pascal создавался как язык для обучения программированию и, поскольку он оправдал ожидания, возникла идея прежде всего сократить длительность цикла обработки студенческих задач. Для этого Виртом было выделено подмножество Pascal-S языка Pascal, достаточное для вводного курса. На языке Pascal был написан аппаратурно-независимый компилятор этого подмножества, пригодный для реализации на самых минимальных вычислительных устройствах. Исходный текст был опубликован в [4], а также детально прокомментирован в [5].

В настоящее время Pascal-S доступен и в электронной форме, в частности, с разрешения Н. Вирта фирма Visible Software распространяет Pascal-S среди своих пользователей. Во время работы компилятор транслирует программу с языка Pascal-S на псевдокод (P-код), представляющий собой машинный язык (ассемблер) некоего гипотетического компьютера, часто

называемого псевдомашиной. Для P-кода просто сделать интерпретаторы, работающие на реально существующих машинах, он легко транслируется в объектный машинный код. Его использовали и создатели UCSD-Pascal в Калифорнийском университете города Сан-Диего. В докладе Вирт указывает на огромную роль, которую сыграла идея P-кода, обеспечившего переносимость и распространение UCSD-Pascal на целый ряд микрокомпьютеров (в том числе машины на базе Intel-8080 и LSI-11 фирмы DEC). UCSD-Pascal имел ряд существенных расширений по сравнению с классическим вариантом языка. В числе наиболее важных стоит отметить [6]:

- строковый тип данных STRING и операции над данными этого типа: сцепление строк, поиск подстроки в строке, удаление подстроки, вставка строки в другую и так далее;
- оператор exit;
- новый тип текстовых файлов INTERACTIVE;
- процедуру Close для закрытия файлов;
- режим произвольного доступа для работы с файлами.

Примером транслятора UCSD-Pascal, распространяемого в настоящее время, является компилятор, входящий в состав Power System (P-система — интегрированная среда для разработки и исполнения программ) британской фирмы Cabot Int. Software, Ltd.

Краткий экскурс в предысторию создания Pascal, предпринятый нами вслед за Виртом, показывает, что Pascal возник не на пустом месте, многие идеи были унаследованы этим языком у своих предшественников. Такая же преемственность сохраняется между Pascal и многими из появившихся впоследствии языков программирования и их диалектов. И тут мы наталкиваемся на классификационную проблему: что считать новым языком (не Pascal), а что рассматривать как диалект (расширение, модификацию) Pascal. Возможная классификация по названию представляется слишком формальной и в ряде случаев просто бессмысленной: «Modula не Pascal» — хорошо, но как быть с модульным Turbo Pascal? Или вслед за автором [7] обозвать его Пасдулой?

Сам Вирт, говоря о будущем языка, отмечает ошибочность некоторых концепций Pascal, что удалось преодолеть в последующих разработках — Modula-2 и Oberon. «Назови я эти языки Pascal-2 и Pascal-3..., их эволюционная линия была бы очевидна» [1]. В свою очередь заметим, что это не единственная эволюционная ветвь, так, в 1989 г. DEC и Olivetti был разработан объектно-ориентированный язык Modula-3 [8], [9] с очень похожим на Pascal синтаксисом. То же можно сказать и о ряде других языков программирования, по сути дела, выражение «Pascal-like syntax» («паскалеобразный синтаксис») стало в настоящее время терминологическим.

Ответить на вопрос, что же такое «истинный» Pascal, призваны стандарты, сыгравшие значительную роль в развитии и распространении этого языка.

языки

По мнению Никлауса Вирта, ошибочность некоторых концепций языка Pascal удалось преодолеть в его потомках — языках Modula-2 и Oberon.

Стандарты языка Pascal

Как известно, стандартизация в области программного обеспечения всегда представляла и представляет собой комплекс трудноразрешимых проблем. Недаром ведущие специалисты в этой области отмечают, что источник трудностей стандартизации кроется в самой природе объекта «программное обеспечение», слишком «абстрактной и гибкой» для уверенной стандартизации [10]. Если рассмотреть проблему более узко, применительно к языкам программирования, то обнаруживаются дополнительные трудности. Так, например, в [11] «указывается на принципиальную неформализуемость современных языков программирования», в результате «противоречия, неясность и неполнота определений языков программирования характерны и для их стандартов (вплоть до лучших международных). Более того, традиционный идеал точного определения языков программирования заводит стандартизацию в тупик...»

Выход автор [11] видит во введении стандартов «с дефектами»...

Итак, специалисты, сознавая недостижимость идеала, все же ищут и находят оригинальные практические решения. К сожалению, существует и другое распространенное мнение, согласно которому стандартизация программного обеспечения вообще не нужна, и даже приносит вред, являясь источником бессмысленных ограничений, тормозя прогресс. Эти мнения с завидным постоянством высказываются, начиная с времен Algol-60 и Fortran вплоть до настоящего времени, применительно к языкам четвертого поколения [10]. На подобные рассуждения анархо-программистов там же, в [10], дается остроумный ответ: даже великие поэты добровольно шли на ограничения, в частности, Шекспир использовал стандарт сонета. Коммерческое программирование, равно как и непроизводственные сферы (наука, образование и так далее), не может обойтись без стандартов на языки программирования, несмотря на всю неидеальность стандартов.

Переходя непосредственно к стандартам языка Pascal, нужно отметить и тот факт, что разные языки программирования по-разному поддаются стандартизации. В частности, это обусловлено аппаратно-зависимыми особенностями того или иного языка. Pascal создавался как язык абстрактной Pascal машины, не привязанной к какой-либо конкретной компьютерной платформе (IBM, DEC, CDC и так далее), что повлекло за собой лавину обвинений в «отсутствии средств ввод-вывода», но зато обеспечило сравнительно легкую стандартизацию Pascal. Другим, не менее важным для стандартизации свойством, явилась простота этого языка — малый словарь ключевых слов и малый объем авторского описания языка [2].

Первые предложения [12] по стандарту языка Pascal были опубликованы в начале 1979 г. рабочей группой Британского института стандартов (BSI), созданной в 1977 г. Эти предложения легли в основу двух эквивалентных стандартов: британского [13] и международного [14]. Незначительно отличается от

них американский стандарт [15]. Впоследствии К. Йенсен и Н. Вирт пересмотрели свое сообщение [16] с учетом международного стандарта.

Международный стандарт [14] предусматривает два уровня языка — 0 и 1, различающихся тем, что уровень 1 допускает гибкую схему передачи массивов как параметров вызова процедур/функций (conformant arrays, или в переводе — «совмещаемый массив-параметр» [17], «согласованный параметр-массив» [18]). Листинг 1 иллюстрирует применения этой схемы. Для испытания программы мы воспользовались компилятором Extended Pascal фирмы Visible Software (режим компиляции — «уровень 1 языка Pascal»). Интересно, что для переноса такой тривиальной программы в Turbo Pascal версии 6.0 ее пришлось существенно модифицировать (листинг 2: измененные и добавленные строки отмечены знаком {+++}). При этом у нас возникли существенные ограничения на размер векторов. Попробуйте, например, обеспечить работу Turbo Pascal процедуры MaxMember для векторов типа

```
vn: array [-30000..-29993] of real;
vp: array [29993..30000] of real;
```

поскольку в ответ на определение

```
vector=[-MaxInt+1..MaxInt] of real;
```

Turbo Pascal возмущается: «Слишком большой массив!». В то же время аналогичную процедуру MaxMember в программе ConformantArrayParameters для векторов vn, vp переделывать не нужно. Заметим, что в новой, седьмой версии Turbo Pascal указанный недостаток преодолен [19].

Pascal, отвечающий международному стандарту [14], часто называют стандартным, или классическим. Однако совсем недавно появился новый стандарт на расширенный Pascal — Extended Pascal [20]. Extended Pascal является расширением стандарта ANSI [15] и полностью совместим с ним (то есть дополняет, но не переопределяет его). Это значит, что программа, написанная в соответствии со стандартами [13]-[15], будет правильной с точки зрения стандарта Extended Pascal при условии, что в ней не применены идентификаторы, соответствующие ключевым словам стандарта Extended Pascal.

К основным расширениям Extended Pascal относятся следующие [20].

1. Модульность. Каждый модуль экспортирует один или более интерфейсов, содержащих значения, типы, схемы (schemata), переменные, процедуры и функции из этого модуля в пределах видимости данного модуля. Переменная может быть защищенной при экспорте, в этом случае импортер (другой модуль или программа) может использовать, но не изменять ее значение. Тип может быть ограничен, в этом случае его структура будет невидима. Имеется возможность переименования идентификаторов при импорте/экспорте. В каждом модуле могут быть описаны действия по его инициализации и завершению работы с ним (finalization).
2. Схемы (schemata) определяются как коллекции подобных типов.
3. Запрос типа параметра.
4. Строки и работа с ними.
5. Механизм связывания переменных (binding), позволяющий установить связь между переменной и внешним окружением: файлом, таймером, командной строкой и так далее.
6. Файлы прямого доступа.
7. Симметрическая разность множеств; функция card, возвращающая число элементов множества; специальный вид оператора for для перечисления членов множества.
8. Арифметика комплексных чисел.
9. Защита параметров процедуры или функции от их изменения внутри этой процедуры (функции).
10. Возведение в степень и вычисление степени числа.
11. Спецификация границ интервалов с помощью выражений.

Наиболее непривычными расширениями, требующими дополнительных пояснений, для большинства читателей, по-видимому, покажутся расширения 2 и 3 приведенного списка — схемы и запросы типов параметров.

С идеей схем можно познакомиться на примере определения нескольких одномерных массивов (векторов) с разным числом элементов, в нашем примере это векторы с двумя, тремя и пятью элементами:

```
type
  VIndex=1..maxint;
  Vector (len: VIndex) =
      array [1..len] of real;
var
  V1, V2: vector (2);
  V3: vector (3);
  V4: vector (5);
```

Отметим, что похожий механизм применяется в Turbo Pascal для описания строк:

```
var
  s1: string [3];
  s2: string [5];
```

Запрос типа параметра в Extended Pascal позволяет определить переменную внутри процедуры того же типа, что и тип параметра этой процедуры:

```
procedure pc(var M: vector);
var
  N: type of M;
```

В заключение рассказа об Extended Pascal упомянем его компиляторы для MS-DOS, выпускаемые двумя фирмами: Visible Software и Prospero Software.

Жизненность любого стандарта определяется его воплощениями. Насколько же конкретные реализации языка Pascal отвечают стандарту? На этот вопрос по отношению к IBM PC, работающим под управлением MS-DOS, отвечает исследование, проведенное в Британском институте стандартов М. Хортоном [21], в котором с помощью специального набора тестовых программ было опробовано семь наиболее распространенных в тот момент компиляторов на соответствие их классическому Pascal [13], [14]. В число отобранных вошли Oregon Software Pascal-2 версия 2.1m, Prospero Software ProPascal версия iid 3.1, Prospero

Software PC Pascal версия 1.1, Borland Turbo Pascal версия 5.0, Microsoft Pascal версия 4.0, Metaware Professional Pascal версия 2.7, Visible Software Dr. Pascal версия 2.04.

Тесты, которым подвергались компиляторы, разбиты на восемь классов, и результаты тестирования сведены в таблицу [21], где указывается число удачных и неудачных тестов определенного класса для каждого из компиляторов. При этом отмечается, что только в компиляторе Oregon Software Pascal-2 предусмотрено соответствие уровню 1 стандарта (согласованные параметры-массивы), кроме того, первые три из перечисленных компиляторов прежде тестировались с помощью более ранней версии набора тестов, поэтому общее количество тестов для разных компиляторов немного различается.

Отмечается, что Turbo Pascal, Microsoft Pascal и Metaware Pascal показали самые плохие результаты по соответствию стандарту. На этом выводы, сделанные в [21], заканчиваются. Однако если проанализировать представленную там информацию, подсчитав процентное отношение количества испытаний, выдержанных каждым компилятором, к общему количеству испытаний, которым он был подвергнут, мы получим, что лучшие результаты показал Dr. Pascal (94%), а худшие Turbo Pascal (61%). Это и неудивительно, если учесть, что стратегия развития Dr. Pascal, предназначенного в первую очередь для целей образования, и реализованного, помимо варианта для MS-DOS, также в вариантах для OS-2 и VAX/VMS, заключается в максимальном следовании стандарту, тогда как Turbo Pascal, созданный исключительно для MS-DOS, ориентируется на максимальное использование системных и аппаратурных возможностей с использованием всевозможных расширений и языковых усовершенствований, нередко явно противоречащих стандарту.

Несмотря на свою нестандартность, а может, и благодаря ей, в настоящее время Turbo Pascal является наиболее распространенным из всех Pascal для MS-DOS. Высказывается даже мнение, что он сам фактически стал стандартом, однако всерьез говорить о приведении языка Pascal к стандарту Turbo Pascal, по-видимому, невозможно из-за жесткой ориентации последнего на MS-DOS и IBM PC. Камнем преткновения в такой стандартизации явилась бы и объектная ориентация поздних (начиная с 5.5) версий Turbo Pascal, поскольку многие «объектно-ориентированные» эксперты отказывают объектно-ориентированному Pascal в праве на существование, говоря, что это дурно понятое ООП. В тоже время другие ехидно замечают им, что спорить об истинном ООП — все равно что выяснять, какая религия истинна, а какая — ложна.

Возвращаясь к испытаниям Хортона, отметим также и тот факт, что ни один из испытуемых не показал 100% соответствия стандарту. Хотя, конечно, какие бы то ни было убедительные выводы из этого сделать трудно. Здесь можно критиковать как испытуемых, так и сам стандарт, а также и методику испытаний.

Следует сказать, что со времени испытаний некоторые фирмы из «Великолепной семерки» утратили интерес к Pascal под MS-DOS. Так, например, новый Oregon Pascal-2 сегодня выпускается американской фирмой TauMetric Corp. для VAX/VMS, Sun-3 и Unix/386, а поставки для MS-DOS прекращены. Затерялся в тумане и Quick Pascal фирмы Microsoft с момента столкновения с Borland Turbo Pascal 5.5, когда на рынке программного обеспечения одновременно появились два объектно-ориентированных Pascal компилятора двух ведущих в мире MS-DOS фирм (эта коллизия была описана в нашумевшей статье Джона Удела, постоянного обозревателя журнала Byte [22]). К сожалению, ответ на запрос редакции «Монитора», адресованный в Microsoft, также не рассеял тумана... Возможно, правомерно утверждение PC Week о том что Borland Pascal побивает Microsoft [41], может, и правда побил?

Аналогичная ситуация наблюдается и с некоторыми другими поставщиками Pascal. Поэтому можно утверждать, что приведенный список производителей Pascal для MS-DOS в настоящее время значительно сократился. Наверное, это можно объяснить как сильной конкуренцией в сфере Pascal продукции, так и конкуренцией со стороны других языков программирования. Остановимся на ней более подробно.

Конкуренты языка Pascal

Пожалуй, наиболее серьезные — это Modula-2, Ada и, конечно, C.

Почему C? Этим вопросом задается в своем докладе «История C++» [23] на упоминавшейся нами конференции HOPL-II Б. Строуструп и отвечает:

* он гибок, не отягчен плохой наследственностью в смысле ограничений;
* он эффективен, его фундаментальная концепция в том, что «C — низкоуровневый язык» (low level);
* он легко реализуется на новом оборудовании и переносится (как сам язык, так и приложения на этом языке не приходится писать заново, «с нуля»).

«Безусловно, успех операционной системы Unix сам по себе был наиболее важным фактором», — отмечает в своем докладе Д. Ричи [24]...

Интересно, что противники C указывают на те же самые причины его популярности. Это можно видеть хотя бы по дискуссиям в рамках постоянно действующей международной телеконференции I-Pascal@UTFSM.bitnet и закрывшейся летом 1993 г. Pascal-L@YALEVM.bitnet. Вопросы о противостоянии C — Pascal нет-нет, да и возникают в дискуссиях. И, как мы уже сказали, анализ ситуации и перечисление причин успехов C в общем-то совпадает с приведенными ранее. Правда, при этом часто заявляют примерно следующее: «Многие мои коллеги, знающие C и Pascal и зарабатывающие на жизнь программированием на C, охотнее работали бы на Pascal». Высказываются и более оригинальные мнения. Так, ожидается, что в связи со свертыванием программы Звездных войн без работы окажется около 50 тысяч программистов NASA, работавших на языке Ada, и, конечно, они скорее предпочтут переключиться с Ada на Pascal, чем на C, и будут оказывать в этом плане давление на своих будущих работодателей, а значит, популярность языка Pascal возрастет. Кто знает? Пока что это споры о будущем.

По-видимому, не последнюю роль здесь может сыграть и дальнейший прогресс в области объектно-ориентированного подхода: пока что C++ представляется

как базовый язык для таких важнейших документов, как спецификации OMG (Object Management Group — ассоциация виднейших производителей программ и аппаратуры в области объектно-ориентированных технологий) [25]. Однако далеко не бесспорно, что такое привилегированное положение сохранится у C++ и в дальнейшем. Так, по материалам самой OMG можно было бы составить обширный список проблем, сопряженных с применением C++ для объектно-ориентированных технологий, в особенности когда речь идет об открытой архитектуре (Open Computing). Для преодоления этих проблем, с одной стороны (в рамках C++), предлагаются различные инструментальные средства, например, [37], с другой, — иные языки. При этом нужно учесть, что продвижение этих языков осуществляется такими влиятельными членами OMG, как, например, DEC (инструментально-языковая система Trellis) или Microsoft (Visual Basic). Конечно, C++, как и любой другой язык программирования, подвергается резким нападкам со стороны его противников, что можно видеть на примере материалов телеконференции по языку Modula-3 (m3@src.dec.com) или выступлений Б. Мейера, автора объектно-ориентированного языка Eiffel [38], и так далее. Конечно, всю эту критику не назовешь беспристрастной, однако в ходе масштабной пропагандистской компании в пользу C++ заявления, подобные приведенному далее, представляются не лишенными объективных оснований:

«...C++ — это лишь веяние моды, и вскоре должен появиться некоторый новый, по-настоящему объектно-ориентированный язык» [39].

Ожидалось, что большое будущее будет иметь язык Ada, но к настоящему времени надежды эти, по-видимому, оправдались не вполне. Не будем вдаваться в детали — это отдельная большая тема, отметим только, что среди энтузиастов этого языка раздается все больше и больше голосов за его ревизию, в частности, за наделение языка Ada более богатыми объектно-ориентированными возможностями.

Ранее говорилось, что, по мнению Вирта, принципиальные недостатки языка Pascal преодолеваются в его дальнейших разработках — Modula-2 и Oberon. Однако в теоретическом плане, если основное отличие языка Modula-2 от Pascal Вирта в модульности, то непонятно, чем Modula-2 лучше Extended Pascal. В плане практической популярности также можно констатировать, что сегодня Pascal, C и Ada — значительно более распространенные языки, чем Modula-2. Что касается языка Oberon, внесшего в ряд «виртовских языков» объектно-ориентированный подход, то за стены лабораторий он пока, как говорится, не вышел.

В теории конкуренция между языками программирования способствует прогрессу и появлению новых более совершенных языков, поэтому споры о том, «какой язык лучше», наверняка небесполезны. Но на практике Вавилонское столпотворение среди программистов порождает множество проблем, одной из которых является перевод исходных текстов с одного языка высокого уровня на другой язык высокого уровня.

На конференции I-Pascal@UTFSM очень часто задается вопрос об инструменте, позволяющем перевести программу, написанную на C, на Pascal, и наоборот — с Pascal на C. Несмотря на сообщения о некоторых shareware («условно-бесплатных») продуктах, доступных через FTP-сервис, по-видимому, удобного инструмента такого типа сейчас просто нет. Более того, в ряде публикаций [44]-[46] о проектах, связанных с переносом с одного языка на другой, говорится о том, что полностью автоматизированный перевод в общем случае вряд ли возможен. Поэтому имеет смысл говорить лишь о полуавтоматическом трансляторе. Такие трансляторы известны, например, для перевода Turbo Pascal 3.0 в Turbo Pascal 5.0 (утилита UPGRADE). Ряд промышленно выпускаемых трансляторов предназначен для перевода с одного диалекта Pascal на другой. Примером такого инструмента для перевода с Turbo Pascal на UCSD Pascal является утилита PascalLiberator фирмы Cabot.

Наверное, в дальнейшим появление и совершенствование инструментов, облегчающих общение между разноязыкими программистами, может оказать большое влияние на языковые рейтинги: с одной стороны, имея необходимое средство, проще не только перейти на новый язык, но и вернуться, отказавшись от не оправдавшего ожидания языка. Хорошую помощь в создании подобных инструментов могут оказать генераторы компиляторов типа YACC, например, PC YACC фирмы Abraxas Software, Inc.

Важным доводом в пользу того или иного языка является ответ на вопрос, насколько серьезные проекты были реализованы на этом языке. Поэтому официальное заявление Ричарда Нелсона (Richard Nelson) из Borland Int., сделанное на телеконференции I-Pascal@UTFSM, в котором подтверждалось ранее опубликованное в PC Week (31 мая 1933) сообщение о том, что Quattro Pro 5.0 для DOS написан на Borland Pascal, заслуживает самого пристального внимания. Нельсон рассказал, что бригада программистов остановила свой выбор на Borland Pascal из-за «производительности (скоросчетности) получаемого кода и продуктивности работы с ним». Он также отметил, что, хотя Quattro Pro for Windows был написан на C++, многие другие серьезные коммерческие программы были написаны на языке Pascal, и назвал ряд таких недавно созданных программ.

Другим важным аргументом в пользу языка программирования является объем литературы, написанной об этом языке (учебники и справочники по языку) и с использованием этого языка (общие и специальные вопросы программирования). И здесь, наверное, языку Pascal нет равных и в области учебников, что понятно, учитывая его происхождение как «языка для обучения программированию», и в сферах общей и специальной литературы, что тоже понятно, если учесть привлекательность гипотетической Pascal машины, позволяющей абстрагироваться от второстепенных деталей, связанных с конкретной аппаратурой. И, напротив, абстрактно описанные алгоритмы очень легко, буквально с листа, реализуются на Pascal. Конечно, первая реализация далеко не всегда оказывается эффективной, но ведь оптимизировать правильно работающую программу зачастую проще и быстрее, чем пытаться сразу написать оптимальную. Итак, заговорив о литературе по языку Pascal, надо отметить и ту ее часть, где формально не используется какой-либо конкретный язык программирования, однако в записи алгоритмов применен Pascal-like syntax, а также синтаксис, легко отображаемый на Pascal (блочная структура, отсутствие goto, составные операторы и так далее).

Литература по языку Pascal

К упомянутой литературе добавим всего несколько названий, в основном, учебников и справочников, которые могут оказаться полезными для практической работы в MS-DOS с современными компиляторами языка Pascal. При этом было бы наивно пытаться в одной статье сделать сколько-нибудь полный обзор даже недавно изданного. Поэтому, увы, многое останется за кадром. С другой стороны, представляется неразумным не упомянуть какую-либо книгу только потому, что она была издана несколькими годами раньше, хотя и не утратила своей актуальности и сегодня.

Книга П. Грогоно [26] явилась первым переводным учебником на русском языке и, возможно, лучшим до сих пор. Особо следует отметить такие моменты, как последовательность и полноту изложения материала, абстрагирование от ограничений конкретных реализаций, хороший выбор примеров, достаточно сложных, чтобы не быть игрушечными, но не слишком длинных, а потому легко доступных для восприятия в деталях. В книге можно найти и обширную библиографию ранних изданий по языку Pascal. В качестве другого библиографического источника, содержащего и более поздние ссылки, можно упомянуть публикацию [27].

Возможно, на чей-то взгляд книга Грогоно покажется слишком толстой, поэтому можно назвать и более краткие учебники по языку Pascal. Наиболее тонким из них является, пожалуй, [28], где предлагается сокращенный вариант языка Pascal, неоспоримым достоинством которого является то, что «всякая программа, правильно написанная на сокращенном Pascal, будет правильной в смысле полного Pascal». Книга предназначена для учащихся средних школ, ПТУ, техникумов и для начинающих программистов.

Кстати сказать, на методической идее минимизации подмножества Pascal, достаточно полного и функционально завершенного для целей обучения с применением принципа «от простого к сложному», построен учебник [29]. Переходя к более поздним учебным изданиям, можно отметить книги «повышенной наглядности», например, [30] и учебник, ориентированный в первую очередь на непрофессиональных программистов, участников «новой компьютерной революции, которая затронет среднюю школу, экономику и быт», — [31].

В книге [31] много хороших примеров, в качестве наиболее интересных можно назвать программу упрощения булевых выражений (с. 147), которую учащийся может легко наращивать и совершенствовать самостоятельно, и программу поиска оптимального маршрута в лондонском метро, которой иллюстрируются многие полезные принципы организации работы программиста. При всем том, что книга Форсайта адресована непрофессионалу и написана хорошим популярным языком, в ней выдержан высокий профессиональный уровень, поэтому ее стоит рекомендовать и начинающим профессионалам, в особенности для самообразования.

В отличие от учебников, делающих упор на выработку правильного стиля программирования, ряд публикаций ориентирован на конкретные версии и реализации языка. К их числу относится уже упоминавшийся учебник по UCSD-Pascal [32]. Информацию о TopSpeed языках (и, в частности, о TopSpeed Pascal) английской фирмы Jensen & Partners Int. можно получить из статьи [42], кроме того, в этом же номере «Монитора» публикуется статья А. Тихонова, целиком посвященная языку TopSpeed Pascal. Уже упоминавшийся здесь Dr. Pascal был описан в [43].

Много книг посвящено различным вопросам программирования на Turbo Pascal. Так, несколько лет назад фирма Borland Int. разослала своим пользователям список, датированный октябрем 1990 г., который содержал более восьмидесяти названий. Помимо учебников и изданий «повышенной наглядности» [33], список содержал и книги, ориентированные на более опытных читателей. В их числе книги N. Shammas (некоторые в соавторстве) — [34], [3].

Ряд книг по Turbo Pascal был написан за последние годы и по-русски. Среди них хочется отметить книгу Е.А. Зуева [35], в которой автору удалось систематически и лаконично изложить довольно объемный материал по особенностям языка Turbo Pascal версии 6.0. Книга состоит из двух частей — «Ядро языка» и «Расширения языка», при этом во вторую часть входят следующие главы: «Модули», «Средства ООП», «Системно-зависимые расширения» (абсолютная адресация, оверлеи, прерывания) и «Программирование на низком уровне» (inline оператор и встроенный ассемблер). Благодаря такой последовательной организации книгу можно читать целиком как учебник и в то же время использовать как справочник. По этой причине книга Зуева не утрачивает своей ценности и для тех, кто собирается изучить следующие за 6.0 версии — Turbo Pascal 7.0 и Borland Pascal. В конце книги приводится список литературы, содержащий 47 ссылок. Что касается новой, седьмой версии Turbo Pascal, упомянем еще раз пособие [19] — можно только удивляться, как двум авторам и фирме «Диалог-МИФИ» в столь короткий срок удалось издать такую хорошую и нужную книгу. Что касается Borland Pascal with Objects версии 7.0 (Turbo Pascal 7.0, поставляемый также и отдельно, фактически является подмножеством этого пакета), упомянем статью [40].

Говоря об учебной литературе, необходимо отметить не только учебники, но и задачники. Из недавно вышедших можно указать [36]. Книга адресована учащимся и основана на опыте работы Литовской заочной школы молодых программистов. Задачи хорошо подобраны: здесь и сортировка методом «пузырька» (в книге почему-то «пузыря»!), и частотный словарь, и восемь ферзей, и числа Фибоначчи, и решето Эратосфена, и игра «Жизнь», и так далее. Практически все они сопровождаются хорошо объясненным решением в виде листингов Pascal программ. ■

5 ноября 1993 года Минюстом РФ зарегистрирован профессиональный Союз программистов (ПСП) России.

Всех, кто заинтересован во вступлении в Союз, в ознакомлении с конкретными проектами, осуществляемыми профсоюзом, и в реализации этих проектов, просьба обращаться в АО «Новые технологии» по адресу:

109180 Москва, 2-й Казачий пер., д. 9, строение 2, кв. 8.

Телефон 238-85-31.

Литература

1. N. Wirth. Recollections about the Development of Pascal// ACM SIGPLAN Not.– 1993.– Vol. 28.– № 3.
2. К. Йенсен, Н. Вирт. Паскаль. Руководство для пользователя и описание языка.– М.: Финансы и статистика, 1982.
3. N. Shammas. Object-Oriented Programming with Turbo Pascal.– NY, 1990.
4. N. Wirth. Pascal-S: A subset and its implementation.– Wiley, 1981.– (Pascal — The language and its implementation/ D.W. Barron, ed.).
5. M. Rees, D. Robson. Practical Compilling with Pascal-S.– Addison, 1988.
6. О.Н. Перминов. Предисловие к книге «Программирование на языке Паскаль».– М.: Мир, 1987.– (Д. Прайс. Программирование на языке Паскаль. Практическое руководство).
7. Opaque Types and Generic Facilities: Modula-2 vs. Turbo Pascal 5.0// J of Pascal, Ada & Modula-2.– 1990.– Vol. 9.– № 1.
8. L. Gardelli, a.o. Modula-3 Report (revised), Nov. 1, 1989// USA: DEC, Systems Research Center.– 1989.
9. S. Harbison. Modula-3. A practical and predictable OOP language for team projects// Byte.– Nov. 1990.
10. B.L. Meek. Problems of Software Standartization// Copmuter Standards and Interfaces.– 1990.– Vol. 10.– № 1.
11. В.М. Кауфман. Частичная формализуемость языков программирования: причины и следствия// Программирование.– 1989.– № 4.
12. A.M. Addyman, a.o. The BSI/ISO Working Draft Standard Pascal// Pascal News.– 1979.– 14 January.
13. BS 6192. Specification for Computer Programming Language Pascal.– London: BSI, 1982.
14. ISO 7185-1982. Specification for Computer Programming Language Pascal.– 1982.
15. American National Standard: Pascal Computer Programming Language, ANSI/IEEE 770x3.97– 1983.– NY, 1983.
16. К. Йенсен, Н. Вирт. Паскаль: руководство для пользователя.– М.: Финансы и статистика, 1989.
17. О.Н. Перминов. Язык программирования Паскаль: Справочник.– М.: Радио и связь, 1989.
18. В. Тумасонис, В. Дагене, Г. Григас. Паскаль. Руководство для программиста: Справочник/ Пер. с литовск.– М.: Радио и связь, 1992.
19. А. Епанешников, В. Епанешников. Программирование в среде Turbo Pascal 7.0.– М.: Диалог-МИФИ, 1993.
20. ANSI/IEEE Standard for Programming Language Extended Pascal, ANSI/IEEE 770X3, 160-1989.
21. M.J. Horton. A Validation Report for Seven Pascal Compilers for IBM PC and Compatibles under MS-DOS.– UK: BSI, 1989.
22. J. Udell. Clash of the Object-Oriented Pascals// Byte.– July 1989.
23. B. Stroustrup. A History of C++: 1979-1991// ACM SIGPLAN Notices.– March 1993.– Vol. 28.– № 3.
24. D. Ritchie. The Development of the C Language// ACM SIGPLAN Notices.– March 1993.– Vol. 28.– № 3.
25. The Common Object Request Broker: Architecture and Specification. OMG Document Number 91.12.1, Revision 1.1. (OMG and X/Open: 1991).
26. П. Грогоно. Программирование на языке Паскаль.– М.: Мир, 1982.
27. M.K. Sadagopan. Pascal Computer Language// J. Instn. Electronics & Telecom. Engrs.– 1983.– Vol. 29.– № 9.
28. С.А. Абрамов, Е.В. Зима. Начала программирования на языке Паскаль.– М.: Наука, 1987.
29. M. Kennedy, M.B. Solomon. PASCAL Program Development with Ten Instruction. Pascal Subset (TIPS) and Standard Pascal.– New Jersey: Prentice-Hall.– 1982.
30. Д. Алкок. Язык Паскаль в иллюстрациях.– М.: Мир, 1991.
31. Р. Форсайт. Паскаль для всех.– М.: Машиностроение, 1986.
32. Д. Прайс. Программирование на языке Паскаль. Практическое руководство.– М.: Мир, 1987.
33. P. Schlieve. Illustrated Turbo Pascal.– 3rd ed.– Wordware Publishing.
34. K. Weiskamp, N. Shammas, R. Pronk. Turbo Algorithms. A programmer's Reference.– J.Wiley & Sons, Inc., 1989.
35. Е.А. Зуев. Язык программирования Turbo Pascal 6.0.– М.: Унитех, 1992.
36. В.А. Дачене, Г.К. Григас, К.Ф. Аугутис. 100 задач по программированию.– М.: Просвещение, 1993.
37. R. Sessions. Developing with IBM's System Object Model (SOM)// First Class, The Object Management Group Newsletter.– 1993.– Vol. 2.– Issue 6.
38. B. Meyer. Ensuring Strong Typing in an Object-Oriented Language. OOPSLA'92// ACM SIGPLAN Not.– 1992.– Vol 27.– № 10.
39. Е. Лепехин. Чем Си++ хуже Си// PC Magazine/ Russian Edition.– 1993.– № 2.
40. В.В. Фаронов. Borland Pascal 7.0: новые возможности// Мир ПК.– 1993.– № 1.
41. P. Coffee. Borland's Pascal Beats Microsoft's for OOP// PC Week.– 1989.– July 3.
42. А.Ф. Яковлев. TopSpeed: вчера, сегодня, завтра// Мир ПК.– 1991.– № 8.
43. М.И. Трофимов. Как увидеть программу?// Мир ПК.– 1992.– № 2.
44. V.D. Moynihan, P.J.L. Wallis. The Design and Implementation of a High-Level Language Converter// Software — Practice and Experience.– 1991.– Vol. 21.– № 4.
45. W.G. Auld, D.J. Goldy. Porting Pascal// J. of Pascal, Ada & Modula-2.– 1989.– Vol. 8.– № 4.
46. K. Bothe, a.o. A Portable High-Speed Pascal to C Translator// ACM SIGPLAN Not.– 1989.– Vol. 24.– № 9.

Листинг 1.
Гибкая схема передачи массивов (уровень 1 стандарта языка Pascal)

```
program  ConformantArrayParameters
         (input, output);
var
   v1 : array [1..3] of real; {первый вектор}
   v2 : array [55..56] of real;{второй вектор}

function MaxMember   (v : array
                        [i1..i2 : integer]
                        of real) : integer;
{Возвращает индекс максимального элемента
вектора, определенного как
   v : array [i1..i2] of real,
где i1,i2 : integer; }

var
   i, m : integer;

begin
   m := i1;
   for i := i1+1 to i2 do
      if v[m]<v[i]
         then m := i;
   MaxMember := m;
end; {_____________ MaxMember _____________}

begin
   v1 [1] := 1.0;
             {инициализируем первый вектор}
   v1 [2] := 3.0;
   v1 [3] := 2.53;
   writeln (MaxMember (v1));
             {печатаем индекс максимального
             элемента первого вектора}
   v2 [55] := 2.0;
             {инициализируем второй вектор}
   v2 [56] := 44.0;
   writeln (MaxMember (v2));
             {печатаем индекс максимального
             элемента второго вектора}
end.
```

Листинг 2.
Альтернативная программа в Turbo Pascal

```
program TurboAlternative (input, output);
var
   v1 : array [1..3] of real;
   v2 : array [55..56] of real;

function MaxMember   (var v: i1,i2 : integer) :
                     integer;              {+++}
type                                       {+++}
   vector = array [0..1000] of real;       {+++}
var
   i, m : integer;

begin
   m := 0;                                 {+++}
   for i := 1 to i2-i1 do                  {+++}
      if vector (v)[m] < vector (v)[i]     {+++}
         then m := i;
   MaxMember := m+i1;                      {+++}
end; {_____________ MaxMember _____________}

begin
   v1 [1] := 1.0;
   v1 [2] := 3.0;
   v1 [3] := 2.53;
   writeln (MaxMember (v1,1,3));           {+++}
   v2 [55] := 2.0;
   v2 [56] := 44.0;
   writeln (MaxMember (v2,55,56));         {+++}
end.
```


Лучшие цены за лучшее оборудование

Для эффективной работы необходимо использовать качественное и надежное оборудование. Компания Демос совместно с австрийскими фирмами **APS-COM** и **FORTUNA** предлагает:

- компьютеры от 386SX до Pentium фирм Hewlett-Packard и DEC
- лазерные принтеры Hewlett-Packard и Canon
- компьютеры-блокноты
- компьютеры класса booksize
- стримеры емкостью от 125Мбайт до 8Гбайт
- цветные и черно-белые сканеры
- графопостроители
- самые дешевые сетевые адаптеры Ethernet, проверенные с Novell NetWare, TCP/IP, DEC Net, MS Windows for Workgroup, NetBIOS
- любое оборудование и инструменты для создания локальных сетей
- настольные издательские системы
- автоматические переключатели для использования одного принтера несколькими компьютерами
- разнообразные принадлежности для компьютеров — звоните

Сегодня мы поставляем только лучшее оборудование, делая все для снижения его цены.

HEWLETT PACKARD Authorized Dealer

digital

Canon

demos™

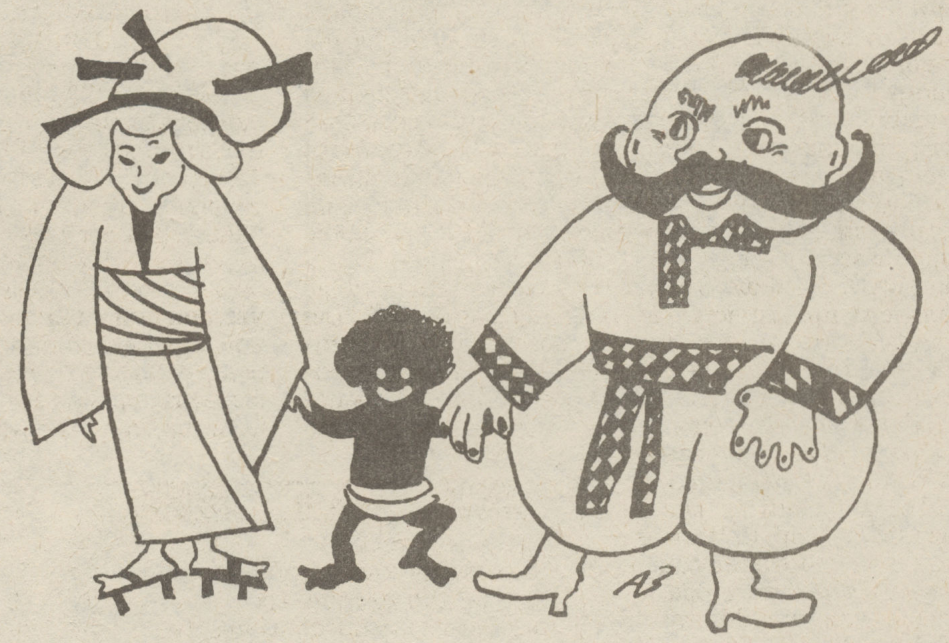
APS-COM

Телефоны: (095) 231-21-29, 231-63-95, 233-05-92, 233-02-42, 231-60-02
Факс: (095) 233-50-16
E-mail: info@hq.demos.su

Знак и логотип **demos** — торговая марка фирмы Демос

TopSpeed Pascal

А. Тихонов



Первоначальной целью статьи являлся только TopSpeed Pascal (TSP), но оказалось невозможным рассматривать систему программирования без анализа концепций TopSpeed (TS) и среды разработки, поэтому начнем сначала — с истории.

Немного истории

Фирма Jensen & Partners International (JPI) была основана Нильсом Йенсеном (N. Jensen), программистом, стоявшим вместе с Ф. Каном у истоков фирмы Borland. С этим связано то, что многие «домашние заготовки» компиляторов обеих фирм имеют одни и те же источники. В отличие от Borland, TS с самого начала ориентировалась не на «широкие круги компьютерной общественности», а на профессиональных программистов. Это наложило отпечаток на все продукты TS: оптимизированный программный код, продуманность состава утилит и интегрированной среды, лаконичность документации, возможность создания программных систем, записанных в ПЗУ и работающих без операционной системы.

Стартовав в 1987 году с компилятора и среды разработки Modula-2, JPI захватила хорошо вычисленную экологическую нишу на рынке инструментальных средств программирования. Modula-2 (M2), разработанная автором языка Pascal Н. Виртом, была ориентирована на создание больших модульных программных систем. Самим Н. Виртом с сотрудниками на M2 была написана операционная среда, включая операционную систему, графический пользовательский интерфейс, компиляторы для рабочей станции Lilith. В отличие от Pascal, в M2 имеется продуманный аппарат раздельной компиляции, экспорта и импорта ресурсов, квазипараллельного исполнения программ [1]. В середине 80-х годов в бывшем СССР имела широкое хождение система программирования M2 для ЭВМ фирмы DEC и СМ ЭВМ и прекрасно работавшая на ДВК-2 в 64 Кбайтах оперативной памяти.

Одной из основных «изюминок» программных продуктов, получивших торговую марку TopSpeed, явилось выделение из компилятора генератора объектного кода, что позволило реализовать под «единой крышей» семейство систем программирования M2, Pascal, C, C++.

В начале 1990 года фирма Clarion, столкнувшаяся с серьезными трудностями при модернизации своей СУБД Clarion, приобрела лицензию на оптимизирующий генератор кода TS [2]. Последующее сотрудничество при разработке Clarion 3.0x привело в сентябре 1992 года к слиянию фирм JPI и Clarion. В настоящее время объединенная фирма поддерживает версию 3.1 своих компиляторов.

Модульность

Первое, что бросается в глаза при знакомстве с TS — последовательно реализованный принцип модульности, начиная со структуры программного продукта, состоящего из модуля среды, компиляторов, инструментальных средств. Модульность распространена и на структуру цены: каждый модуль стоит 99$. Одну и ту же цену имеют модули среды для DOS (реального и защищенного режима), OS/2, языковые процессоры, инструментальные подсистемы (TS TechKit) и небольшой пакет программ для создания СУБД.

Модульность сказывается и в структуре программного кода. Модуль исходного текста состоит из двух файлов — интерфейсного модуля, содержащего объявления данных, процедур, списки импортируемых ресурсов, и модуля реализации, включающего в себя тексты процедур, описания внутренних для данного модуля данных. Файлы интерфейсных модулей имеют расширения def для M2, itf для Pascal,h для C и C++. Модуль реализации может отсутствовать, но в этом случае ему должен соответствовать объектный файл или библиотека.

Поддержка различных платформ

Вторым принципом реализации всех компиляторов семейства TS является поддержка различных плат-

форм. Работая, например, в DOS или OS/2, программист может легко сгенерировать программу для реального режима DOS, защищенного режима DOS, Windows 3.x, OS/2. Указание целевой платформы осуществляется переключателем в интегрированной среде, либо командой в файле проекта. Кроме этого, объектный модуль может быть помещен либо в обычную библиотеку объектных модулей, либо в так называемую динамическую библиотеку DLL (Dynamic Link Library). Степень поддержки платформ в TS неодинакова. Наиболее полным является обеспечение реального, привычного для всех нас режима DOS. Для защищенного режима имеется собственный 16-битовый расширитель DOS (TS DOS Extender). При программировании для Windows 3.x и OS/2 программист во многом предоставлен самому себе. Так, все сведения по программированию для Windows ограничиваются 20 страницами в руководстве по TS TechKit. Несмотря на существование собственного TS компилятора ресурсов, наличие Microsoft Software Development Kit (SDK) представляется необходимым.

ЯЗЫКИ

Последовательно реализованная модульность, поддержка различных платформ и многоязычность — вот основные принципы семейства компиляторов TopSpeed.

Для OS/2 в TS поставляются интерфейсные модули для использования библиотеки функций (API) операционной системы и Presentation Manager; за подробностями программист отсылается к руководствам по OS/2.

Многоязычность

Третий принцип TS — многоязычность: поддерживаются M2, Pascal, C, C++, ассемблер, представляющие собой единое семейство. Программный код, написанный на различных языках TS, хорошо совместим даже на уровне объектно-ориентированных расширений.

Компоненты TS

Минимальная конфигурация, необходимая для работы с TS: модуль среды и один из компиляторов (M2, Pascal, C, C++). Таким образом минимальная стоимость системы составляет 198$. Среда TS поставляется в следующих разновидностях: реальный режим DOS, защищенный режим DOS, OS/2. Для профессиональной работы необходим модуль TS TechKit. Поставляются исходные тексты всех библиотек TS. Заслуживают упоминания библиотека API расширителя DOS, библиотеки, предназначенные для создания баз данных и телекоммуникационных программ. Установка любого компонента очень проста. В процессе работы над статьей выяснилось, что ненужные компоненты необходимо деинсталлировать, так как при любом изменении среды автоматически призводится реконфигурация системы. Простое стирание программ делает систему неработоспособной. Здесь же отметим компактность TS, так, совместно установленные на винчестере TS Pascal и M2 занимают 5,8 Мбайтов, а C++ около 8 Мбайтов.

Поддержка разработки больших программных систем

Создатели TS много сделали для поддержки работы над большими программными проектами. Рассмотрение этого аспекта TS мы начнем с системы проектов. TS проект — вещь ненавязчивая и обнаруживается только в случае крайней необходимости. Так, при начале работы с программой система сообщает об отсутствии файла проекта и предлагает завести новый, после чего опять становится невидимой. И только при увеличении сложности разрабатываемой программной системы обнаруживаешь, что система проектов поддерживается специальным языком, сравнимым по выразительным возможностям с другими языками семейства TS и значительно превышающим по мощности и удобству make и другие средства обеспечения разработки больших программ, имеющиеся в UNIX или в системах программирования семейства Borland. Часть информации о проекте может быть изменена из интегрированной системы без всякого редактирования файла проекта.

При разработке больших программ с помощью компилятора гипертекстовой информационной системы можно дополнять и изменять встроенную систему помощи сведениями о собственных библиотеках, интерфейсах процедур и функций. Компилятор гипертекстовой информационной системы входит в состав TS TechKit.

Особого упоминания заслуживают средства выполнения больших программ, не помещающихся в опера-

БИБЛИОТЕКИ DLL

Динамические библиотеки

TS является единственной известной мне системой программирования, позволяющей использовать DLL (Dynamic Link Library) в реальном режиме DOS. Рутинная процедура создания исполняемого модуля состоит в компиляции исходного текста программы в один или несколько объектных файлов с последующей обработкой объектного кода редактором связей. Редактор связей компонует объектные файлы программы с библиотечным объектным кодом системы программирования в COM или EXE файл, настраивает адреса. Весь программный код должен присутствовать в исполняемом модуле.

При использовании DLL редактирование связей осуществляется «на лету», то есть библиотечный модуль подгружается в оперативную память и настраивается только в тот момент, когда он вызван другим модулем. Это позволяет поместить общий для некоторого семейства программ код в одну библиотеку DLL на диске. Кроме этого, для платформ, допускающих мультипрограммный режим, например, OS/2 или Windows 3.x, это позволяет экономить оперативную память за счет загрузки в память только одного экземпляра библиотечного модуля, используемого несколькими программами. Так, практически все ядро OS/2 и Windows 3.x — это динамические библиотеки.

тивной памяти. К ним относятся оверлейная система TS и виртуальная память, работающая в защищенном режиме под управлением расширителя DOS. Для использования оверлеев программисту достаточно либо в файле проекта, либо в интегрированной оболочке указать модель памяти overlay, а все остальное сделает TS. Оверлейная система работает не только с сегментами программного кода, но и со статическими данными. Все это похоже на средства, имеющиеся в Turbo Pascal (TP), но, в отличие от TP, в TS имеется мощная библиотека для управления оверлейной системой. Программист может указать, какие процедуры и данные должны быть загружены в память при запуске программы, какие фиксированы в оперативной памяти, какие выгружены на магнитный диск, а какие не нуждаются в выгрузке.

Расширитель — новая компонента TS, появившаяся только в версии 3.1. Для его использования также достаточно указать соответствующую модель памяти.

Расширитель DOS функционирует на ПЭВМ с процессорами 80286 и выше в 16-битовом защищенном режиме, позволяя адресовать до 16 Мбайтов оперативной памяти. Выполняя программу в защищенном режиме, расширитель при обработке прерываний и вызове функций DOS временно переходит в реальный режим. Кроме этого, расширитель DOS поддерживает работу с виртуальной памятью объемом до 512 Мбайтов. В DOS Extender ToolKit имеется библиотека, позволяющая программисту управлять всеми средствами расширителя и виртуальной памяти.

Виртуальная память дает возможность автоматически выполнять программы, требующие для себя и своих данных больше оперативной памяти, чем имеется в наличии. Виртуальная память работает как с программным кодом, так и с динамическими и статическими данными. Для повышения эффективности работы расширитель использует информацию о частоте обращения к сегментам программного кода и данных, находящихся в оперативной памяти. Редко используемые сегменты вытесняются из оперативной памяти первыми. Так как сегменты с программным кодом загружаются в оперативную память только тогда, когда они нужны, то программы, созданные для выполнения в защищенном режиме с использованием виртуальной памяти, начинают работать значительно быстрее, чем обычные программы.

В разработке больших программ существенно помогает собственная TS-система редактирования связей, которая проверяет интерфейсы обращения к процедурам и функциям, исключая тем самым большинство ошибок, встречающихся при программировании на C, исключает из исполняемого модуля неиспользуемые код и данные. В последних версиях системы это свойство распространяется и на объектно-ориентированные языковые расширения: из исполняемого модуля исключаются как неиспользуемые статические, так и виртуальные методы. Такая возможность весьма существенна при работе с большими библиотеками классов.

Опираясь на собственные силы

Еще одной существенной чертой семейства программных продуктов TS является то, что в КНДР в свете идей Чучхе называется опорой на собственные силы. В TS все свое — от собственных форматов объектного кода, системы проектов, редактора связей, расширителя DOS до компиляторов ресурсов для Windows, гипертекстовой информационной системы и средства для сжатия исполняемых модулей.

В TS имеется такое экзотическое средство отладки, как возможность создания и анализа посмертного дампа программы, которое, как мне казалось, кануло в Лету вместе с ОС ЕС. Оказалось, что, в отличие от ЕС ЭВМ, дамп можно распечатать в удобочитаемой форме, анализ же его осуществляется в терминах исходного текста программы с помощью штатного интерактивного отладчика. Этот «пережиток» может сэкономить много времени и нервов, когда у пользователя появляется трудновоспроизводимая ошибка в программе. Забрав автоматически созданный посмертный дамп, можно в комфортных домашних условиях проанализировать его и отыскать причины, приведшие к ошибке.

В TS TechKit содержатся уникальные средства, которые в других системах программирования являются самостоятельными продуктами. К ним относится перехватчик прерываний DOS Watch, дисассемблер, средства создания TSR-программ, работы с последовательными портами, собственный профайлер и быстрый однопроходный ассемблер. Следует также отметить полный набор средств для создания встроенных программ, записываемых в ПЗУ и работающих без поддержки операционной системы.

Собственный редактор связей и общий для всех компиляторов генератор кода позволяют использовать нестандартные средства оптимизации кода. Так, кроме стандартных соглашений, используемых в C и Pascal, о передаче параметров процедурам через стек, TS поддерживает собственное соглашение о связях, в котором передача параметров осуществляется через регистры. В ряде случаев, особенно при наличии значительного числа коротких подпрограмм с небольшим числом параметров передача их через регистры может значительно повысить скорость выполнения программы. Выбор соглашения о связях осуществляется либо из интегрированной оболочки, либо включением соответствующих псевдокомментариев (прагм) в файл проекта. В TS широко используется развитый аппарат прагм, то есть инструкций, включаемых в исходный текст программы и/или файл проекта, и предназначенный для управления генерацией кода и обеспечения совместимости программ, разработанных на различных языках программирования.

Интегрированная оболочка

Десятиоконная интегрированная оболочка (ИО) TS достаточно традиционна и похожа на интегрированную оболочку TP версии *5.5*. Однако в отличие от последней она поддерживает «мышь», а также ряд простых, но очень нужных программисту утилит, которые не мешало бы включить и в другие системы программирования. К ним относятся простенький калькулятор, позволяющий включать результаты вычислений в текст программы, ASCII-таблица, мощные средства контекстного текстового поиска в файлах. Думаю, что любой программист, работавший с программной системой, состоящей более чем из двух десятков файлов, оценит возможность найти нужный исходный текст, не выходя из ИО.

Интерактивный отладчик

VID (Visual Interactive Debugger) позволяет отлаживать многоязычные программы TS в терминах исходного текста, на уровне ассемблера на всех поддерживаемых платформах, включая анализ посмертных дампов и дисассемблирование фрагментов исполняемого модуля.

VID поддерживает большое число типов контрольных точек: постоянные, однократные, вызывающие останов при выполнении некоторого условия, отложенные (аналогичные однократным, но вызывающие останов после выполнения данного оператора программы заданное число раз).

VID обеспечивает пошаговое выполнение программы, пропуск процедур, переход к текущему положению курсора в исходном тексте программы, к заданной программистом процедуре, к выходу из текущей процедуры, к ближайшей контрольной точке и так далее.

В интерактивный отладчик встроены средства для работы с выражениями. Выражения могут включать в себя идентификаторы переменных, арифметические и логические операции, операции с указателями. Во время отладки можно получить вычисленное значение выражения, наблюдать изменение его значения в процессе выполнения программы, остановить выполнение программы, когда данное выражение станет истинным.

В VID встроено большое число средств, обеспечивающих просмотр и модификацию значений переменных, регистров, стека вызовов подпрограмм. Имеются средства отслеживания переменных, получивших или изменивших значения в каждый момент выполнения программы.

VID позволяет вести отладку на одном или на двух дисплеях одновременно.

Особенности входного языка TSP

После затянувшегося вступления обратимся к цели статьи TopSpeed Pascal. Реализация TSP фактически содержит три входных языка: во-первых, язык, соответствующий международному стандарту ISO 7185 Level 1, во-вторых, собственно TS расширения Pascal, и, в-третьих, средства, обеспечивающие совместимость с Turbo Pascal (TP).

TSP, в отличие от TP, ставшего фактическим стандартом, полностью поддерживает стандарт ISO, включая массивы и строки с открытыми границами (здесь и далее будет использоваться терминология TP 7.0 и [3]). В стандарте данное свойство названо conformant arrays.

Поддержка стандарта ISO стала хорошим тоном для компиляторов Pascal, разработанных в последние годы, примером тому может служить Prospero Extended Pascal фирмы Prospero Software.

Далее мы рассмотрим наиболее существенные отличия TSP от TP.

В TSP, как и в стандарте языка, необходимо указывать имена файлов стандартного ввода-вывода в заголовке программы:

```
PROGRAM P(INPUT,OUTPUT);
```

При запуске программы файл INPUT будет автоматически открыт оператором RESET, а OUTPUT — оператором REWRITE. Стандартный ввод-вывод легко перенаправить, указав в командной строке соответствующие им файлы операционной системы.

Правила задания констант для записей и массивов заимствованы из Modula-2.

```
CONST
  MaxAge = 120;
  MaxSize = 2;

TYPE
  PERSON = RECORD
      Name: STRING[40]; { имя }
      Age: 1..MaxAge;     { возраст }
  END;
  COUPLE = ARRAY[1..MaxSize] OF PERSON;
                                { семейная пара }
  VECTOR = ARRAY[1..MaxSize] OF INTEGER;
                                { вектор }
  MATRIX = ARRAY[1..MaxSize] OF VECTOR;
                                { вектор }

CONST
  M = MATRIX(VECTOR(1,2),VECTOR(2,1);
  PARTNERS = COUPLE(PERSON('Jack Ripper',34),
  PERSON('Red Hat',16));
```

Так же как и в TP, в TSP допустимо объявление переменных, фиксированных в оперативной памяти:

```
VAR
  TIMER[40H<<4+6CH] OF INTEGER;
                          { четырехбайтовое целое }
```

Из переменной TIMER всегда можно получить текущее время в единицах, кратных 54,93 миллисекундам (шестнадцатеричные константы завершаются символом H, а сдвиг влево сегментной части адреса осуществляется оператором <<).

Нововведением TSP является возможность назначения псевдонимов для переменных. Это удобно, например, в случае длинных имен полей записей.

```
TYPE
  COUPLE = RECORD { пара }
      H: PERSON;
      W: PERSON;
  END;

VAR
  FAMILY_COUPLE : COUPLE; { семейная пара }
  ALIAS                   { псевдонимы }
      HUSBUND = FAMILY_COUPLE.H;{ муж }
      WIFE = FAMILY_COUPLE.W;{ жена }
```

TSP и TP (с точностью до названий) оперируют с одними и теми же скалярными типами данных, соответствия между которыми приведены в таблице.

В дополнение к обычным указателям в TSP введены базированные указатели, сегментная часть адреса которых связана с некоторым сегментом. Это позволяет легко работать с перемещаемыми в оперативной памяти структурами, например, создавать собственные диспетчеры памяти. Переменные типа базированного указателя содержат только смещение, сегментная

Таблица

Скалярные типы TSP и TP

Тип TSP	Тип TP	Описание
CHAR	CHAR	Символьный тип
BOOLEAN	BOOLEAN	Однобайтовый булевский тип
INT8	SHORTINT	Однобайтовый целый тип, значения в диапазоне -128..127
BYTE	BYTE	Однобайтовый целый тип, значения в диапазоне 0..255
INT16	INTEGER	Двухбайтовый целый тип, значения в диапазоне -32768..32767
WORD	WORD	Двухбайтовый целый тип, значения в диапазоне 0..65535
INTEGER	LONGINT	Четырехбайтовый целый тип, значения в диапазоне -2147483648..2147483647
FIXREAL	COMP	Восьмибайтовый тип с фиксированной точкой, значения в диапазоне -9.0E18..9.0E18
SHORTREAL	SINGLE	Четырехбайтовый тип с плавающей точкой, значения в диапазоне 1.2E-38..3.4E+38, точность 6-7 десятичных знаков
REAL	DOUBLE	Восьмибайтовый тип с плавающей точкой, значения в диапазоне 2.3E-308..1.7E+308, точность 15-16 десятичных знаков
LONGREAL	EXTENDED	Десятибайтовый тип с плавающей точкой, значения в диапазоне 3.4E-4932..1.1E+4832, точность 19-20 десятичных знаков

часть адреса берется из описания переменной (листинг 1).

В листинге 1 использован оператор :>, означающий передачу битового образа без преобразования и контроля корректности. Этот оператор используется в основном для присваивания значений указателей разных типов и адресной арифметики.

Введен оператор :: преобразования одного типа в другой.

```
TYPE Colors = (Red, Green, Blue);
...
(2:: Color)      { возвращает Blue}
(1:: BOOLEAN){ возвращает TRUE}
(65::CHAR)     { возвращает 'A'}
```

При задании прагмы check(range=>on) осуществляется контроль корректности преобразовании типа с помощью оператора ::.

Для оптимизации программного кода в TSP описанию процедуры и функции может предшествовать ключевое слово INLINE. Это означает, что вызов процедуры при компиляции автоматически заменяется на текст ее тела.

Как было сказано ранее, TSP поддерживает массивы и строки с открытыми границами:

```
PROCEDURE F(VAR A:ARRAY[LOW..HIGH:INTEGER]
                    OF CHAR);
  VAR I :INTEGER;

BEGIN
  FOR I:= LOW TO HIGH DO A[I] := I;
END;
```

В заголовке процедуры для формального параметра A задается тип индекса массива INTEGER, а также его нижняя и верхняя границы. При вызове процедуры параметрам LOW и HIGH автоматически присваиваются нижняя и верхняя границы массива — фактического параметра. В теле процедуры границы массива ведут себя так, как будто это локальные константы, объявленные в процедуре.

В заголовках процедур допустимо использование нетипизированных параметров.

```
TYPE
  STRING80 = STRING[80];
  CHARARRAY80 = ARRAY[1..80] OF CHAR;

PROCEDURE T(A:INTEGER; VAR RESULT);
BEGIN
  CASE A OF
     0: (RESULT:>STRING80) := 'STRING80';
     1: (RESULT:>CHARARRAY80) := 'CHARARRAY80';
  END;
END;

VAR
  S: STRING80;
  C: CHARARRAY80;

BEGIN
  T(0,S); WRITELN(S);
                { выводит на экран: STRING80 }
  T(1,C); WRITELN(C);
                { выводит на экран: CHARARRAY80 }
END.
```

При вызове процедуры в нее автоматически передается информация о размере нетипизированного параметра, которая может быть получена внутри нее с помощью функции SIZE. Передачу этой информации можно подавить с помощью прагмы. Нетипизированные параметры используются для обеспечения совместимости с другими языками семейства TS, где параметр может не иметь фиксированного типа.

Раздельная компиляция

В международном стандарте отсутствует механизм, обеспечивающий раздельную компиляцию частей программы. Вся программа должна быть в одном файле, никаких библиотечных функций, кроме предопределенных в стандарте, не допускается.

Для обеспечения раздельной компиляции модулей и работы с библиотеками в TS введен единый для всех языков механизм, обеспечивающий проверку интер-

фейсов в уже скомпилированных и компилируемых модулях.

Каждый модуль делится на две части-файла: интерфейсную и реализационную.

Интерфейсный модуль содержит объявления констант, псевдонимов, типов, переменных, заголовки процедур и функций, которые могут экспортироваться в другие модули.

Объявления в реализационной части модуля являются внутренними по отношению к нему. Интерфейсная часть модуля имеет расширение ITF, реализационная — PAS. Реализационная часть модуля должна иметь соответствующую ей интерфейсную часть. Обратное необязательно — интерфейсной части может соответствовать объектный модуль, библиотека объектных файлов, DLL-библиотека. В этом случае в интерфейсном модуле необходимо задать прагму module(implementation=>off).

Далее приведены интерфейсная и реализационная части модуля DATE. Файл DATE.ITF:

```
INTERFACE UNIT DATE;
CONST
  First_year = 1980;
  Last_year = 1999;

TYPE
  WeekDay = (Sun, Mon, Tue, Wed, Thu, Fri, Sat);
  DayRange = 1..31;
  MonthRange = 1..12;
  YearRange = First_year..Last_year;
END.
```

Реализационная часть файла DATE.PAS имеет вид:

```
IMPLEMENTATION UNIT DATE;
  ...
BEGIN
  ... { инициализация модуля }
END.
```

Модули могут импортировать типы, переменные, процедуры и функции из других модулей, для чего их имена должны быть перечислены в секции объявлений IMPORT. Например, при использовании средств программирования квазипараллельных процессов необходимо вставить в модуль секцию

```
IMPORT PasProc;
```

Это обеспечивает «ограниченный» импорт, то есть при использовании ресурсов необходимо явно указывать имя модуля, из которого осуществляется импорт, например, PasProc.InitSignal (Finish).

Можно избежать явного указания имени модуля, используя так называемый глобальный импорт, для чего в секции IMPORT после имени модуля необходимо указать *:

```
IMPORT PasProc *;
```

TSP допускает импорт отдельных ресурсов модуля, для чего импортируемые ресурсы перечисляются в скобках через запятую сразу после имени модуля. Эти ресурсы можно использовать без указания имени модуля.

Объектно-ориентированное расширение TSP

Принципы реализации объектно-ориентированного расширения TP и TSP в основном совпадают и имеют своим источником Object Pascal [4]. Основными отличиями являются множественное наследование, а также то, что в TSP все методы по умолчанию являются виртуальными, отсутствуют специальные конструкторы и деструкторы.

Описание класса, как и модуля, разделено на интерфейсную и реализационную части. В листинге 2 приведены интерфейсная и реализационная части класса SongsAndDances, наследующего свойства классов Songs и Dances.

Выше уже говорилось, что по умолчанию методы являются виртуальными, для объявления их статическими необходимо перед именем метода указать ключевое слово STATIC.

Реализационная часть объявления класса заключена в операторные скобки IMPLEMENTATION OF Имя_Класса; ... END;

Если класс объявлен в интерфейсной части модуля, то его реализационная часть должна находиться в реализационной части модуля. Там же должны быть интерфейсные и релизационные части классов, предназначенных только для внутреннего использования в модуле.

Множественное наследование порождает необходимость разрешения конфликта между совпадающими именами полей и методов базовых классов. В TSP эта проблема решается следующим образом. В производном классе поля или методы с одинаковыми именами становятся невидимыми, обращение к ним возможно только при явном указании имени класса, например, Songs.Activity.

Еще одним способом решения данной проблемы является использование аппарата псевдонимов, как показано в листинге 2. Заметим, что для объявления псевдонимов не используется секция ALIAS.

В объектно-ориентированном расширении TSP введен оператор IS, позволяющий во время исполнения программы проверить принадлежность объекта данному классу в процедуре или функции, формальный параметр которой является базовым классом или указателем на базовый класс.

Поясним это на примере. Пусть базовым классом будет FirstClass; ему наследует SecondClass, а ThirdClass является наследником SecondClass.

```
...
FUNCTION Test(VAR C:FirstClass): BOOLEAN;
BEGIN
  Test := (C IS SecondClass);
END;

VAR
  C2: SecondClass;
  C3: ThirdClass;
  B: BOOLEAN;

BEGIN
  B:= Test(C2); { B получит значение TRUE}
  B:= Test(C3); { B получит значение FALSE}
END;
```

Функция Test осуществляет проверку принадлежности к классу SecondClass.

Действие оператора преобразования :: можно использовать и для преобразования полей объектов. Если преобразование некорректно, то может возникнуть ошибка во время выполнения. Проиллюстрируем сказанное примером.

```
CLASS C1;
  X: INTEGER;
END;

CLASS C2 INHERITS C1;
  Y: INTEGER;
END;
...

PROCEDURE A(VAR C:C1);
BEGIN
  IF C IS C2 THEN BEGIN    { проверка типа }
     X:= 200;
     (C::C2).Y := 300; { преобразования поля Y
                         формального параметра
                         класса C1 к C2 }
     END
  ELSE C.X := 100;
END;
```

Вместе с TSP поставляются исходные тексты библиотеки классов, содержащей средства работы со списками, коллекциями, стеками, очередями, векторами, матрицами, двоичными деревьями (включая сортировку), хэш-таблицами, потоками в оперативной, дополнительной памяти и на диске.

Кроме этого, реализован класс обработки исключительных ситуаций в стиле языка программирования Ada, включающий в себя средства их возбуждения, обработки, а также рестарта программы.

Листинги библиотеки классов служат примером хорошего стиля объектно-ориентированного программирования и читаются с пользой и удовольствием.

Библиотечные модули TSP

PasDos обеспечивает интерфейсы с DOS: работу с прерываниями операционной системы, директориями, командной строкой, окружением DOS; обработку критических ошибок программистом. Кстати, последние средства реализованы наилучшим образом среди известных мне систем программирования.

Модуль PasLib реализует часть стандартной библиотеки Pascal, а также собственные расширения TSP, включая средства ввода-вывода, управления памятью, вычислений с плавающей точкой, работы со строками и генерации случайных чисел.

Модуль PasRed позволяет использовать специфические для TS средства работы с файловой конфигурацией прикладной программной системы. Конфигурационный файл, легко модифицируемый пользователем, позволяет связать файлы с определенными расширениями с директориями на диске. Кроме этого, модуль обеспечивает выделение из полного имени файла пути и расширения.

В модуле PasWin приводится мощная объектно-ориентированная система для работы с текстовыми окнами, в том числе и с перекрывающими друг друга, включая в себя открытие, закрытие, ввод-вывод, изменение размеров окна, цвета текста, рамки, заголовка.

В TSP включены модули, обеспечивающие совместимость с TP по библиотечным процедурам и функциям. К ним относятся TurboSys, TurboCrt, Graphics. Последний требует наличия библиотеки Graphics.Lib Turbo C, демонстрируя тем самым совместимость двух этих систем программирования.

Использование библиотек других языков программирования TS не вызывает затруднений. TS поставляет интерфейсные модули для их взаимного использования. Так, в листинге 3 приведена программа, использующая библиотечный модуль GRAPH Modula-2 для работы с 256-цветовой графикой. Данная программа рисует 256-цветовую «таблицу настройки» с умалчиваемой палитрой, затем та же картинка показывается с палитрой, состоящей из «чистых» красного, зеленого и синего цветов, наконец, демонстрируется картинка со случайной серой палитрой. Переход от одной стадии программы к другой осуществляется, когда пользователь нажимает любую клавишу.

Кстати, я с недоумением обнаружил, что функция RemapPalette, отвечающая за изменение входов палитры в режиме 256 цветов, работает неправильно.

Библиотека Modula-2 значительно богаче TSP: из нее можно импортировать средства для работы с дополнительной памятью, «мышью».

Квазипараллельные процессы

Отдельного разговора заслуживает модуль PasProc, обеспечивающий квазипараллельное исполнение до 32 процессов. Последнее ограничение может быть снято перекомпиляцией исходных текстов модуля PasProc.

Обеспечение параллельного исполнения нескольких программ является неотъемлемой частью развитых операционных систем, начиная с ОС ЕС. Такие средства имеются в Unix и OS/2. Windows/NT поддерживает не только квазипараллельное исполнение процессов на одном процессоре, но и параллельную работу программ на нескольких процессорах.

В DOS отсутствуют средства поддержки параллельных процессов, кроме этого, реализация квазипараллельности затруднена из-за нереентерабельности собственных программ DOS.

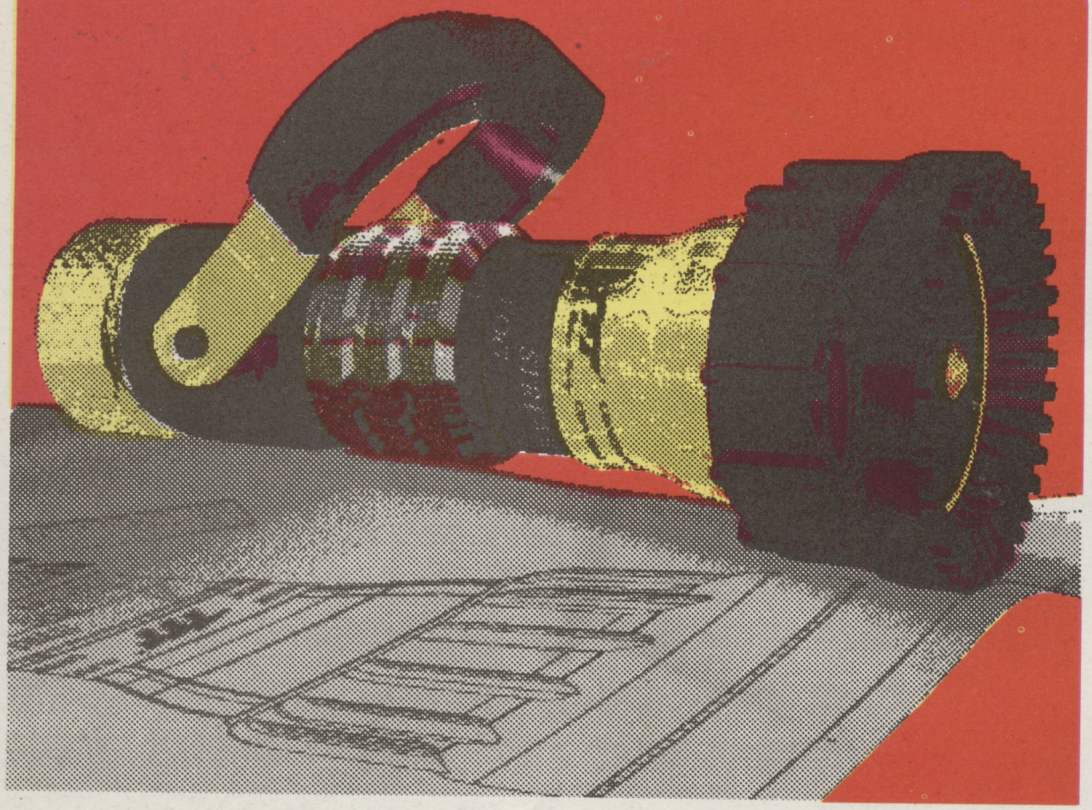
Квазипараллельное программирование в TS осуществляется предоставлением каждому из выполняющихся процессов кванта времени, после чего процесс прерывается, его состояние, включая состояние регистров и стека, сохраняется, управление передается планировщику TS, который запускает следующий процесс, стоящий в очереди на выполнение.

Планировщик управляет процессами с учетом их приоритетов, то есть процессы с одинаковым приоритетом запускаются в порядке очереди, но процесс с низким приоритетом может получить квант времени только при отсутствии готовых к выполнению процессов с более высоким приоритетом. В TS наивысшим приоритетом считается нулевой, чем менее важен процесс, тем больше номер его приоритета.

Процессы взаимодействуют друг с другом с помощью сигналов. Сигналы представляют собой очереди сообщений. Каждый сигнал обладает собственным

LVS Corporation. 65 Profsoyuznaya Str, Moscow 117806, Phone (095)330 3733 Fax (095) 330 3693

AUTODESK 3D STUDIO®



RELEASE 3

Более 200 новых возможностей

Тонирование в сети

Autodesk 3D Studio® версии 3 позволяет преодолеть барьер производительности PC. 3D Studio может поручить задачи тонирования нескольким (до 9. 999) компьютерам в локальной сети и автоматически распределить их между процессорами, доступными в данный момент времени в сети.

Качество тонирования на PC как на рабочей станции

В 8 раз повышена четкость изображения; добавлено управление размытыми очертаниями объектов при движении; скорость тонирования выросла на 40%.

Новые эффекты освещения

Проецирование неподвижных или движущихся изображений, как если бы это осуществлялось с помощью слайд- или кинопроектора; эффекты вращения и наезда источников света; Ray-tracing; в процессе анимации источники можно выключить.

В редакторе материалов достижимо качество Truecolor

Управление насыщенностью, яркостью и прозрачностью текстуры материалов путем беления, смешивания или расслаивания. Присвоение особых металлических свойств таким материалам, как бронза, золото или сталь.

Удобства, ускоряющие работу

Контекстно-зависимая подсказка; возможность связать часто используемые команды с функциональными клавишами; тонирование выбранных объектов или зоны экрана; Autodesk 3D Studio® версии 3 способнаработать как полноэкранное DOS-приложение в среде Microsoft Windows.

Профессиональная анимация

Материалы преобразуются в процессе анимации (например, прозрачный пластик превращается в хромированный металл); изменение характеристик источников света в процессе анимации; камеры и источники света могут быть подчинены друг другу или выбранным объектам.

Сочетание Autodesk 3D Studio® версии 3 с Autodesk Animator Pro, Автокадом и Visual Link гарантирует создание высококачественных профессиональных фильмов и неподвижных изображений высокого разрешения.

Приглашаем к сотрудничеству дилеров!

Autodesk.

Телефон: (095) 261-63-63
Факс: (095) 267-99-27

STEEPLER

Телефон: (095) 245-63-36
Факс: (095) 245-63-36

555 666 777 888

Вниманию чековых инвестиционных фондов и крупных акционерных обществ

Финансовый центр "Грант"
представляет программный комплекс

"Реестр акционерного общества"

Технология ФЦ "Грант" обеспечивает решение проблем размещения и учета ценных бумаг акционерного общества.
Используя нашу технологию, руководство акционерного общества ежедневно будет в курсе текущего состояния акционерного капитала.
Возможно, у Вас уже есть мнение об уровне услуг в области фондового рынка, поэтому мы рекомендуем не спешить принимать решение, а ознакомиться с нашей системой.

Технология ФЦ "Грант" позволяет формировать реестр от нескольких сотен до нескольких десятков миллионов акционеров.

Программный комплекс "Реестр акционерного общества" создан на лицензионных инструментальных средствах.

ГРАНТ
GRANT

Контактный телефон: (095) 238-99-33
Факс: (095) 238-32-78

555 666 777 888

Компьютерный «ларек» 7.93

Код	Продукт	Краткая характеристика	Объем поставки	Цена, $	Требования к оборудованию
SL-01	Turbo Games	Библиотека для Turbo Pascal 6.0-7.0. Спрайты, звуковые эффекты, трехмерная графика, стереоэффекты. Доступ к ZIP файлам.	2 дискеты + документация	15	EGA адаптер, IBM PC/AT
SL-02	Data Vision 1.5	Объектно-ориентированная библиотека для Turbo Pascal 6.0-7.0. Создание и управление реляционными базами данных. Поддерживаются DBF и DB форматы, а также электронные таблицы.	1 дискета + документация	35	IBM PC/XT
SL-03	Atoll	Библиотека для Turbo C. Динамический просмотр и масштабирование графических изображений.	2 дискеты + документация	10	IBM PC XT/AT, EGA/VGA, PCX-system
SL-04	SPRight Lib	Объектно-ориентированная библиотека для Borland C++ 1.xx-2.xx. Спрайты, спрайтовые объекты, звуковые эффекты, поддержка графического формата PCX. Прозрачна для большинства графических библиотек.	1 дискета + документация	20	EGA/VGA адаптер, IBM PC/AT
SL-05	Easy Work-3	Библиотека для Quick и Professional BASIC. Интерфейс, управление "периферией", конструктор электронных таблиц.	1 дискета + документация	12	IBM PC/XT
SL-06	MAESTRO	Библиотека для Turbo Pascal и C++. Воспроизведение звуковых эффектов и музыки. Поддержка PC Speaker, Covox (LPT1-LPT3), Sound Blaster (DMA и Direct Mode). Совместимость с синтезатором.	1 дискета + документация	40	IBM PC/AT
SC-01	QuickTools 1.0	Средство разработки диалога в Paradox.	1 дискета + документация	12	IBM PC/AT, EGA/VGA, 3 Мб на HDD, 1-2 Мб ОЗУ.
SC-02	QB Tools 3.0	Средство разработки диалога, а также поддержки DBF формата для Quick и Professional BASIC.	1 дискета	30 для индивидуальных покупателей, 50 для организаций	IBM PC/AT, EGA, 3 Мб на HDD, 640 Кб ОЗУ
SC-03	АвтоГРАФ	Пакет для создания на экране графиков с результатами обработки числовых данных и печати их на 9 и 24-игольчатых принтерах. Поддержка форматов ЕСКД.	1 дискета 360 Кб + документация	10	IBM PC/XT, EGA/VGA, 500 Кб на HDD, 200 Кб ОЗУ
SC-04	ETXT-LQ	Текстовый и графический редактор. Генератор шрифтов. Печать на любых матричных принтерах.	4 дискеты	14	IBM PC/XT, 1,1 Мб на HDD, 384 Кб ОЗУ
SC-05	Сфера 2.1	Создание и ведение справочников. Формирование и заполнение бланков. Форматированная печать договоров.	1 дискета + документация	25	IBM PC/AT 1,5 Мб на HDD, 350 Кб ОЗУ

Если Вас заинтересовали программные продукты или Вы хотите получить дополнительную информацию, звоните, пишите, приезжайте.

Оплата принимается только в рублях по текущему курсу доллара.

Адрес для писем: 117419 Москва, а/я 765.

Тел.: (095) 943-25-53, 238-99-33. Факс: (095) 238-32-78.

именем. Процессы во время исполнения могут посылать и получать сообщения. Передача сообщений осуществляется под управлением планировщика, что сравнительно безопасно, хотя программист может устроить себе веселую жизнь, если несколько процессов будут ожидать сигналов друг от друга.

Не все процедуры могут выполняться квазипараллельно без специальных мер предосторожности. Основным условием является реентерабельность всех прерываемых или перезапускаемых планировщиком процессов. Для преодоления этой трудности в процессе выделяются так называемые критические участки, которые блокируются от прерывания планировщиком. Единственным требованием к критическому участку является то, чтобы он был как можно короче. В виде критических участков оформляются все обращения к DOS (например, ввод-вывод), изменения значений переменных, общих для нескольких процессов, и так далее. После выполнения критического участка необходимо проинформировать планировщик о возможности прерывания данного процесса. Далее планировщик продолжает свою работу по распределению ресурсов между процессами в соответствии с их готовностью и приоритетами.

Все программы, использующие квазипараллельные процессы, должны быть скомпилированы с моделью памяти MThread. Это делается для того, чтобы сгенерировать специальный реентерабельный код, необходимый для работы планировщика.

Структура программы, использующей квазипараллельные процессы, имеет следующий вид. В головной программе должна быть осуществлена инициализация сигналов, глобальных ресурсов (переменных, буферов, окон), разделяемых процессами, запущен планировщик.

После этого в головном модуле запускаются все квазипараллельные процессы и головная программа «засыпает» до получения сигнала окончания работы, обусловливающего закрытие файлов, освобождение памяти и другую работу по аккуратному завершению программы.

Квазипараллельные процессы оформляются в виде процедур, в начале которых осуществляются действия по инициализации. Все основные действия процесса выполняются в бесконечном цикле. В этом цикле процесс ожидает некоторое событие: им может быть нажатие пользователем определенной клавиши клавиатуры или получение сигнала. После получения сигнала процесс выполняет его обработку и опять переходит к ожиданию события.

В отличие от «последовательного» программирования процессы должны ассоциироваться с очень маленькой задачей, например, с получением символа с клавиатуры, выводом его в окно и тому подобным. В листинге 4 приводится программа PROCEXAM, осуществляющая квазипараллельный вывод случайных символов со случайными атрибутами в шесть окон. Выполнение программы прекращается при нажатии любой клавиши клавиатуры. Кроме всего прочего, из этого листинга можно получить некоторое представление о работе с текстовыми окнами в TS.

Создание и особенно отладка параллельных программ — достаточно трудоемкий и антиинтуитивный, во всяком случае для меня, процесс, поэтому существенной оказалась помощь интерактивного отладчика VID, поддерживающего отладку квазипараллельных процессов. Оказалось, что не на всех ПЭВМ с нулевым временем ожидания (к ним относятся 20 и 25 МГц машины с процессором 80286 и 40 МГц с процессором 80386) могут работать квазипараллельные процессы, что, видимо, связано с их аппаратными особенностями. Это доставило много неприятных минут при отладке PROCEXAM, прежде чем я сообразил, что не виноваты ни я и ни TSP, после чего стал пытаться запускать программу на всех попавшихся под руку машинах.

В связи с квазипараллельным программированием хочется упомянуть еще об одном компоненте системы программирования TS: Communication ToolKit, обеспечивающем средства для создания драйверов коммуникационных программ, передачи файлов с помощью протоколов XMODEM, YMODEM, KERMIT, эмуляции терминалов VT52/VT100, описания сценариев сеанса связи. В сочетании с рассмотренными ранее возможностями библиотека может служить основой для создания телекоммуникационного программного обеспечения, в том числе обслуживающего несколько модемов.

Вопросы совместимости с другими системами программирования

Режим совместимости TSP с TP задается прагмой module(turbo_comp=>on). Исходный текст программы на TP должен быть предварительно преобразован программой TP2TS. TS обеспечивает правильный перевод более 90% языковых конструкций TP версий 5 и 6. Не конвертируются встроенный машинный и ассемблерный код, процедуры с описателями INTERRUPT и EXTERNAL, директивы компиляции, отличные от {$I имя_файла}.

Для анализа совместимости программ TP и TSP я конвертировал модуль просмотра графических файлов GIF [5]. Мне хотелось проверить совместимость не только по входному языку, но и по библиотекам системы программирования, особенно по графике, а также проанализировать возможности оптимизации программного кода. Программа размером более 800 строк исходного текста была отконвертирована без ошибок, однако при первом прогоне TSP выдал 36 ошибок. Отладка программы заняла более полутора часов. Наибольшие трудности вызвало то, что если в TSP процедура или функция описана в интерфейсном модуле или с ключевым словом FORWARD, то при ее реализации нельзя указывать список ее параметров. Вторым камнем преткновения явились различия в библиотеках: так, функция InitGraph имеется как в собственном модуле GRAPH TS, так и в модуле GRAPHICS, реализующем BGI графику. Достаточно долго я не мог понять, почему TSP при редактировании связей требует библиотеку GRAPHICS.LIB, которую необходимо было переписать из Turbo C. Во всех перечисленных трудностях я виню себя, надо было более внимательно читать документацию.

В результате моих стараний был создан модуль размером 49 Кбайтов против 28 Кбайтов TP, выполнявшийся на 10% быстрее. Включение всех возможных видов оптимизации TS практически не изменило ситуации. Вспомнив, что в TP версии программы ограничивающим фактором являлся вывод пикселов на

экран, и подтвердив этот факт с помощью собственного средства TS для анализа выполняемой программы (profiler), я попытался использовать библиотечный модуль GRAPH. К сожалению, описание этого модуля содержится только в Modula-2 Library Reference. В исходном тексте программы пришлось изменить только процедуры инициализации графического режима и вывода пикселов. Время выполнения программы сократилось более чем в полтора раза. Это говорит не столько о качестве сгенерированного программного кода, сколько о неэффективности BGI.

Для сравнения эффективности программного кода я написал простую программу умножения двух матриц размерности 80×80, состоящих из десятибайтовых чисел с плавающей точкой (тип EXTENDED TP и LONGREAL TSP). Выборка элементов матриц осуществлялась с помощью процедур, с тем чтобы проверить эффективность обработки вызова процедур. Размеры исполняемых модулей оказались 18320 и 23338 байтов, а время исполнения 19.8 и 11.5 с для TP и TSP, соответственно.

Следует отметить хорошую совместимость внутри семейства компиляторов TS. Библиотеки одного компилятора достаточно свободно могут быть использованы в программах, написанных на другом языке программирования. Имеется совместимость даже на уровне объектно-ориентированных расширений. В качестве примера вместе с TSP поставляется библиотека классов C++, реализующая независимую от оборудования графику, которую легко можно подключить к программам на TSP и Modula-2. Однако для работы с межъязыковыми интерфейсами нужно иметь представление о реализации языковых конструкций, системе прагм TS, изучить соответствующие главы руководств и примеры межъязыковых интерфейсов.

Нет проблем с использованием объектного кода Microsoft C и Turbo C, если вызывающая программа откомпилирована одним из компиляторов TS. Для используемых объектных библиотек необходимо написать интерфейсные модули и правильно указать прагмы. Несколько сложнее приходится, когда головным модулем служит программа на Microsoft C или Turbo C. Для обеспечения совместимости может оказаться необходимым перекомпилировать тексты стандартных библиотек. С помощью такого приема Е. Киселю удалось подсоединить объектные модули TS к программам Arity Prolog.

Документация

TSP поставляется с двумя томами документации — TopSpeed Pascal Language Reference и TopSpeed Pascal Library Reference — и рядом файлов документации на магнитных носителях. Сразу отмечу, что название «справочник» достаточно точно отражает характер документации. Она написана ясным языком, но не помешали бы дополнительные примеры использования языковых конструкций, библиотечных процедур и функций.

Language Reference посвящен особенностям входного языка TSP, отличиям от международного стандарта, вопросам преобразования программ, написанных на TP в TSP. Вторая книга описывает библиотечные модули TSP.

Вместо заключения

Программирование — это прежде всего ремесло; производительность и качество работы программиста зависят от полноты набора и качества инструментов, с которыми он работает. Если сравнивать программирование в средах DOS, Windows, OS/2 на Modula-2, Pascal, C, C++ с работой по дереву, камню и металлу, то TS представляет собой единый, удобный, настраиваемый под индивидуальные особенности набор инструментов, рассчитанный на профессионала, знающего все приемы работы с такими различными материалами.

В статье в основном говорилось об уникальных особенностях TS. Если бы статья была написана года полтора назад, на этом можно было бы закончить. Однако теперь необходимо сказать и о том, чего нет в TS: средств конструирования пользовательских графических интерфейсов и визуального программирования в средах Windows и OS/2, имеющихся в Borland Pascal, C++, Visual C фирмы Microsoft, Symantec C++ 6.0 (бывший Zortech). Будем надеяться, что все это и многое другое появится в новой версии 32-битовых компиляторов, к выпуску которой готовится фирма Clarion Software.

Мне осталось поблагодарить сотрудника фирмы INSIGHT А. Пуховского и сотрудников фирмы АЛКОНС С. Горина, А. Чернышова за предоставленные материалы, консультации и обсуждение. ■

Литература

1. Э. Неппли, Р. Платт. Программирование на языке Модула-2.– М.: Радио и связь, 1989.
2. Б.Д. Баррингтон. Как создавался Кларион// Мир ПК.– 1993.– № 2.
3. К. Йенсен, Н. Вирт. Паскаль. Руководство пользователя и описание языка.– М.: Финансы и статистика, 1982.
4. Г. Буч. Объектно-ориентированное проектирование с примерами и задачами.– Киев: Диалектика, Москва: ИВК, 1992.
5. А. Тихонов. Показать картинку. Часть 2// Монитор.– 1993.– № 5.

Стопроцентное восстановление качества красящих лент для принтеров на специальном оборудовании без извлечения ленты из картриджа.
Тел. (095) 943-25-53

Редакция журналов «Монитор» и «Монитор-Аспект» приглашает к сотрудничеству специалистов, желающих принять участие в работах по тестированию программных продуктов и аппаратуры.

Обращаться по тел. 335-68-31.
Дмитрий Бройтман

Листинг 1.
Базированные указатели

```
CONST
   PoolSize = 2000;
VAR
   Pool: ARRAY[0..PoolSize] OF BYTE;
   PoolTop: 0..PoolSize;
TYPE
   PoolPtr = ^(SEG(Pool)) PoolRec;
                { так описывается указатель,
                базированный на массив Pool,
                сегментная часть указателя
                вычисляется с помощью
                функции SEG }
PoolRec = RECORD
   Next : PoolPtr;
END;
CONST
   PoolNil = 0:>PoolPtr;
                { так задается нулевое смещение
                относительно базы, с помощью
                оператора преобразования :> }
PROCEDURE ALLOCATE(VAR  P:PoolPtr;
                        ElementSize:WORD);
BEGIN
   P := PoolTop:>PoolPtr;
      { преобразование типа PoolTop к PoolPtr }
   PoolTop :=  PoolTop + ElementSize +
               SIZE(PoolRec);
END;
BEGIN
   ...
   ALLOCATE(P);
   P^.Next := PoolNil;
END.
```

Листинг 2.
Описание класса с множественным наследованием

```
CLASS SongsAndDances INHERITS Songs, Dances;
   ...
   SongsActivity = Songs.Activity;
   DancesActivity = Dances.Activity;
      { описание псевдонимов }
END;  { SongsAndDances }
IMPLEMENTATION OF SongsAndDances;
      { реализация методов }
BEGIN
      { блок инициализации класса }
END;
```

Листинг 3.
Графические средства TSP

```
PROGRAM program256(output);
   IMPORT   GRAPH *  {это библиотека Modula-2},
            PASDOS*, PASLIB *;
TYPE
   Pal256 = ARRAY[0..255] OF
               ARRAY[1..3] OF BYTE;
         { палитра режима 256 цветов}
var
   i,j,k,Col,jr,ic : WORD;
   BB : Boolean;
   C : CHAR;
   w,d,n,PIXEL : word;
   v : VideoConfig;
   { запись с параметрами текущего
   графического режима }
   PR,PCOL : INTEGER;
   P,R,G,B : WORD;
   REG : REGISTERS;
PROCEDURE WRITEPAL(VAR PAL:PAL256);
   { изменение палитры режима 256 цветов }
VAR
   REG : REGISTERS;
BEGIN
   REG.AH := 010H;
   REG.AL := 012H;
   { подфункция 12h прерывания 10h:
      установить несколько регистров
      таблицы цветов (регистров ЦАП)}
   REG.BX := 000H;   { номер первого регистра }
   REG.CX := 256;    { число цветов }
   REG.ES := SEG(PAL[0,1]);
   { сегментная часть таблицы цветов }
   REG.DX := OFS(PAL[0,1]);
   { смещение таблицы цветов }
   INTR(010H,REG);
END;
PROCEDURE TRUECOLORS;   { чистые цвета }
VAR
   PAL : PAL256;
   PIXEL : WORD;
   R,G,B : BYTE;
BEGIN
   for Pixel := 0 to 255 do begin
      P := PIXEL mod 64;
      IF ((Pixel>=0) and (Pixel<64)) then
      BEGIN       { 64 градации серого }
         R := P;  { красный }
         G := P;  { зеленый }
         B := P;  { синий }
      END;
      IF ((Pixel>=64) AND (Pixel < 128)) THEN
      BEGIN
         R := P;
         G := 0;
         B := 0;
      END;
      IF ((Pixel>=128) and (Pixel < 192)) THEN
      BEGIN       {64 градации зеленого }
         R := 0;
         G := P;
         B := 0;
      END;
      IF Pixel >= 192 THEN
      BEGIN       {64 градации синего }
         R := 0;
         G := 0;
         B := P;
      END;
      PAL[Pixel,1] := R;
      PAL[Pixel,2] := G;
      PAL[Pixel,3] := B;
   END;
   WRITEPAL(PAL); {изменяю палитру}
END;
PROCEDURE RANDOMCOLORS;
      { случайное изменение палитры,
      здесь реализованы градации серого }
VAR
   PIXEL,C : WORD;
   R,G,B,M : BYTE;
   PAL : PAL256;
BEGIN
   FOR PIXEL := 0 TO 255 DO BEGIN
      FOR C := 1 TO 3 DO
         PAL[PIXEL,1] := RANDOM(64);
      PAL[PIXEL,2] := PAL[PIXEL,1];
      PAL[PIXEL,3] := PAL[PIXEL,1];
   END;
   WRITEPAL(PAL);
END;
BEGIN
   RANDOMIZE;
      { инициализирую датчик случайных чисел }
   BB := SetVideoMode(_MRES256COLOR);
      { устанавливаю режим 256 цветов }
   GetVideoConfig(v);
      { читаю информацию о графическом режиме}
   N := v.numcolors;    { число цветов }
   W := v.numxpixels;
      { разрешение по горизонтали }
   D := v.numypixels;
      { разрешение по вертикали }
   IF BB THEN BEGIN
      { вывожу цветовую таблицу на экран }
      FOR i := 0 TO W-1 DO
         FOR j := 0 TO D-1 do BEGIN
            ic := i div 20;
            jr := j div 10;
            Col := (jr*16+ic) mod n;
            Plot(i,j,Col);
         END;
      C := RdKey; { жду нажатия любой клавиши }
      TRUECOLORS; { показываю чистые цвета }
      C := RDKEY; { жду нажатия любой клавиши }
      REPEAT
         RANDOMCOLORS;
         { случайная "серая" палитра, пока
         не будет нажата любая клавиша }
      UNTIL KEYPRESSED;
      TEXTMODE; { перехожу в текстовый режим }
   END;
   WriteLn(' Цветовое разрешение:',N);
   WriteLn(' Горизонтальное разрешение:',W);
   WriteLn(' Вертикальное разрешение:',D);
   C := RDKEY;
      { жду нажатия любой клавиши }
END.
```

Листинг 4.
Квазипараллельные процессы

```
PROGRAM ProcExamp(Output);
IMPORT PasProc,PasDos,PasWin,PasLib;
CONST
   TimeDelay = 2;        { задержка }
TYPE
   TitleName = String[40];
VAR
   MainWin : PasWin.PopUpWin;
   Finish : PasProc.Signal;
   FinishFlag : BOOLEAN;
      { флажок завершения процессов }
```

```
PROCEDURE CreateWindow     { создать окно }
                (VAR  W:PasWin.PopUpWin;
                      X,Y,
                   { верхний левый угол окна }
                      Width,Height : WORD;
                   { ширина, высота окна }
                      ForeGround,
                      BackGround : PasWin.Color;
                   { цвета текста и фона }
                      Title : TitleName
                   { имя окна }
                );
CONST
   Shadow = FALSE;
      { тень от создаваемого окна }
   AtTop = TRUE;
      { имя окна на верхней рамке }
BEGIN
   W.Open(X,Y,Width,Height,
        ForeGround,BackGround);
      { открываю окно }
   W.FrameDraw(PasWin.DoubleFrame,ForeGround,
        BackGround, Shadow);
      { вычерчиваю двойную рамку }
   W.FrameWrite(Title, AtTop,
     (Width - Length(Title)) div 2+1,
       ForeGround,BackGround
       );
      { вычерчиваю имя окна }
   W.WrapOn := TRUE;
      { перенос со строки на строку }
END;  {CreateWindow}
PROCEDURE Colors(VAR ForeGround,
        BackGround:PasWin.Color);
      { случайное формирование цветов }
CONST
   ForeColors = 16;
   BackColors = 8;
VAR W : PasWin.PopUpWin;
BEGIN
   PasProc.Lock;
   ForeGround := PasLib.Random(ForeColors):>
                 PasWin.Color;
      { обратите внимание на преобразование
      типа :> }
   REPEAT
      BackGround := PasLib.Random(BackColors):>
                    PasWin.Color;
   UNTIL ForeColors <> BackColors;
   PasProc.Unlock;
END;  {Colors}
PROCEDURE RandomIO(VAR W:PasWin.PopUpWin);
   { вывод случайных символов в текущее окно }
VAR
   N : INTEGER;
BEGIN
   W.PopUp;
      { показываю окно поверх других окон }
   W.MakeFocus;
      { перехожу к окну W }
   FOR N:=0 TO PasLib.Random(100) DO BEGIN
      Colors(W.Fore,W.Back);
      { устанавливаю случайные цвета }
     Write(CHR(32 + PasLib.Random(224)));
   END;
END;  {RandomIO}
PROCEDURE HouseKeeping(VAR
                  W:PasWin.PopUpWin);
VAR
   X, Y,
   Width,
   Height : WORD;
   ForeGround,
   BackGround : PasWin.Color;
   OPENED : BOOLEAN;
BEGIN
   OPENED := TRUE;
   REPEAT
      PasProc.Lock;
      IF OPENED THEN RandomIo(W);
      IF (FinishFlag AND OPENED) THEN BEGIN
         W.Close;
         OPENED := FALSE;
      END;
      PasProc.Unlock;
      PasProc.Delay(TimeDelay);
   UNTIL FALSE;
END;  {HouseKeeping}
{ ************** Процессы **************}
PROCEDURE ToFinish;
{ процесс, следящий за реакцией пользователя}
VAR
   B : BOOLEAN;
BEGIN
   FinishFlag := FALSE;
   REPEAT
      PasProc.Lock;
      B := PasDos.KeyPressed;
      IF B THEN BEGIN
         PasProc.Send(Finish);
         FinishFlag := TRUE;
      END;
      PasProc.Unlock;
      PasProc.Delay(TimeDelay);
   UNTIL FALSE;
END;
PROCEDURE CheckKeyBoard;
{ проверка нажатия клавиши клавиатуры }
BEGIN
   FinishFlag := FALSE;
   REPEAT
      PasProc.Lock;
      FinishFlag := PasDos.KeyPressed;
      PasProc.UnLock;
      IF FinishFlag THEN PasProc.Send(Finish);
      PasProc.Delay(TimeDelay);
   UNTIL FinishFlag;
END;
PROCEDURE P1;
VAR W : PasWin.PopUpWin;
BEGIN
   PasProc.Lock;
   CreateWindow(W, 3, 3, 20, 3,
        PasWin.LightBlue, PasWin.Red, 'P1');
   PasProc.Unlock;
   HouseKeeping(W);
END;  {P1}
PROCEDURE P2;
VAR W : PasWin.PopUpWin;
BEGIN
   PasProc.Lock;
   CreateWindow(W, 60, 12, 15, 8,
        PasWin.White, PasWin.Blue, 'P2');
   PasProc.Unlock;
   HouseKeeping(W);
END;  {P2}
PROCEDURE P3;
VAR W : PasWin.PopUpWin;
BEGIN
   PasProc.Lock;
   CreateWindow(W, 5, 7, 30, 8,
        PasWin.Black, PasWin.Cyan, 'P3');
   PasProc.Unlock;
   HouseKeeping(W);
END;  {P3}
PROCEDURE P4;
VAR W : PasWin.PopUpWin;
BEGIN
   PasProc.Lock;
   CreateWindow(W, 56, 2, 18, 2,
        PasWin.Blue, PasWin.LightGray, 'P4');
   PasProc.Unlock;
   HouseKeeping(W);
END;  {P4}
PROCEDURE P5;
VAR W : PasWin.PopUpWin;
BEGIN
   PasProc.Lock;
   CreateWindow(W, 40, 12, 10, 7,
        PasWin.Black, PasWin.LightGray, 'P5');
   PasProc.Unlock;
   HouseKeeping(W);
END;  {P5}
PROCEDURE P6;
VAR W : PasWin.PopUpWin;
BEGIN
   PasProc.Lock;
   CreateWindow(W, 3, 17, 12, 4,
        PasWin.Blue, PasWin.DarkGray, 'P6');
   PasProc.Unlock;
   HouseKeeping(W);
END;  {P6}
BEGIN
   PasLib.Randomize;
   FinishFlag := FALSE;
   CreateWindow(MainWin,1,1,76,22,
        PasWin.Black,PasWin.LightGray,
        'Основное окно');
   PasProc.StartScheduler;
      { запускаю планировщик процессов}
   PasProc.Init(Finish);
      { инициализирую сигнал }
   PasProc.StartProcess(ToFinish,1000,1);
   PasProc.StartProcess(P1,4000,1);
   PasProc.StartProcess(P2,4000,1);
   PasProc.StartProcess(P3,4000,1);
   PasProc.StartProcess(P4,4000,1);
   PasProc.StartProcess(P5,4000,1);
   PasProc.StartProcess(P6,4000,1);
   PasProc.WAIT(Finish);
      { после нажатия любой клавиши
      управление передается сюда }
   PasProc.StopScheduler;
      { останавливаю планировщик процессов }
   MainWin.Close; { закрываю окно }
END.  {ProcExamp}
```

Когда кругом в дыму снаряды рвутся...

(Картинки с выставки)

Уф... Отмаялись... «SofTool-93» позади. Нет, что ни говорите, но любая большая компьютерная выставка с точки зрения «человека на стенде» чем-то похожа на маленький сумасшедший дом. И надо иметь достаточно крепкую психику, чтобы в течение недели снова и снова объяснять потенциальному покупателю достоинства предлагаемой тобой продукции. А со всех сторон на тебя скалятся злобно улыбающиеся мониторы конкурентов, ненавистными голосами орут всякие CD-ROM и Sound Blaster... А шныряющие в толпе карапузы только и норовят утащить себе что-нибудь на память. Ну как, ощутили? То-то же. Но ведь не бросают они это столь опасное дело, нет, снова и снова возвращаются и принимают участие во всех выставках. И это, господа-товарищи, самая большая из коммерческих тайн проклятых буржуинов. Почему они возвращаются в сутолоку и гам, раздают бесплатно красивые картинки и наклейки, а некоторым счастливчиками — даже пакеты! Но не дождетесь Вы на этот вопрос ответа от Мальчиша-Плохиша — его нынче печеньем да вареньем не купишь! Вот сало, простите, «Сникерс» — это да! Как говорится, съел и свободен! Но разве накупишься проклятых буржуинских шоколадок на подкуп агента?

Задумался читающий народ, пригорюнился. А значит, если дошли Вы до этой точки, то я своей основной цели уже добился. От статьи Вас уже не оторвешь. И это правильно. Для Вас эта статья и писалась. Кому же ее еще читать? А речь в ней пойдет об итогах последней выставки и о глобальной роли «Монитора» в ее освещении.

Итак, обо всем по порядку. Конечно, если сам себя не похвалишь, никто не похвалит. Поэтому начнем, да простят меня организаторы выставки, с самовосхваления. «SofTool» проводится уже не первый год и в рекламе не нуждается. Поэтому мы, естественно, не могли пропустить такое событие. Более того, нас (да и остальных участников и организаторов выставки) не остановила ни стрельба на улицах, ни чрезвычайное положение. Да, 5 октября, в день открытия, посетителей было маловато. Но через пару дней народ отошел от первого испуга и повалил, что называется, валом.

Мы очень благодарны посетителям выставки за высокую оценку нашего журнала. Как догадываются наши читатели, в эпоху рынка голосование «за» и «против» осуществляется очень просто — рублем. В результате спрос резко превысил предложение и мы, к сожалению, не сумели осчастливить всех желающих последними номерами — расхватывали слишком быстро. Но высокая оценка журнала — это не только, да и, прямо скажем, не столько объем денежного мешка. Это, я Вам доложу, отношение людей. И как приятно взглянуть на высокого, откормленного молодого человека в «тройке», который, по-видимому, потратил на девушку последние карманные деньги и с тоской разглядывает разложенные на стенде журналы. И разве начнешь поднимать скандал, хотя видишь, что особо полюбившийся номер «Монитора» тайком прячется в карман? Нет, дорогие читатели, это словами не передать!

Но, конечно, выставка «SofTool-93» не ограничивалась экспозицией нашего журнала. На ней присутствовали свыше 170 фирм, подавляющее большинство из которых — наши, российские. Это очень отрадный факт, особенно если учесть, что значительно снизился процент участников, предлагающих по исключительно низко завышенным ценам «стандартные конфигурации PC/AT» и разный ширпотреб в придачу.

Позвольте кратко остановиться на том, что произвело наибольшее впечатление. **Приводимый в данной статье материал отражает только личную точку зрения автора и не может расцениваться как реклама!**

Во-первых, это продукция фирмы Apple. Фирма RUI собрала на свой стенд практически всех дилеров компьютеров Macintosh и программного обеспечения для них. Посетители проявили живейший интерес, но покупателей, похоже, было не так уж много. Да это и понятно. Самый важный вопрос — где брать математику — пока остается без ответа. В самом деле, для IBM PC совместимых компьютеров программное обеспечение не пишет только ленивый, да и сами PC у нас более распространены. А если и захочешь сотворить что-нибудь для Mac, то где взять справочную литературу? Поэтому, поцокав языком возле шикар-

ного монитора со встроенной стереосистемой (!), редакция решила открыть на страницах нашего журнала рубрику, посвященную практической работе на компьютерах фирмы Apple. Надеемся, что она окажется не бесполезной.

Конечно, сторонники направления PC не могли не ответить на этот вызов. Так, фирма «СофтЮнион» предлагала посетителям законченные издательские линии, вплоть до фотонаборных автоматов AGFA. Надо признать, что комплексность поставки (CD-ROM, магнитооптические накопители большой емкости и так далее) несколько отпугивала потенциальных покупателей. В самом деле, стоимость полного комплекта в 154000 $ заставляет задуматься. Но, надо полагать, он стоит этих денег.

Следующим направлением в расширении возможностей PC совместимых ЭВМ является технология мультимедиа. Рынок отреагировал на робкие обзоры компьютерной прессы с явным опережением. Огромное количество участников представляло те или иные звуковые платы, контроллеры CD-ROM и видеоускорители. Но народ отнесся ко всему этому великолепию очень настороженно. Интерес проявлен, но ценовая политика фирм пока оставляет желать лучшего. Можно ли рассчитывать на массовое внедрение плат, стоимость которых составляет 150-250 долларов и прикладная ценность которых для большинства пользователей ограничивается играми?

Были, конечно, на выставке и более дешевые, доступные платы. Например, фирма «Алькор-Мультимедиа» предлагала звуковую плату для IBM PC, «совместимую с Sound Blaster и Adlib». Но при ближайшем рассмотрении обнаружилось, что FM-синтезатора, являющегося ядром как Adlib, так и Sound Blaster, на плате не видно. Все преобразование цифра-аналог осуществляется с помощью ЦАП 572ПА2. Никаких сигнальных процессоров или заказных БИС на плате обнаружено не было. Говорить о том, что проблемы решаются за счет резидентных программ, нельзя по той простой причине, что программирование плат прикладными программами осуществляется, как правило, через порты ввода-вывода. Так что можно говорить только о совместимости с Sound Blaster в части прямого преобразования цифра-аналог. Музыка и Adlib остаются, увы, за кадром. А жаль, цены были не так уж и высоки!

Но, с другой стороны, может быть, именно игры и явятся тем звеном в цепочке «производитель-потребитель», которое, наконец, приведет к качественному скачку в оснащении персональных ЭВМ. Одна из приятных особенностей выставки состояла в том, что наконец-то компьютерные игры стали одним из самостоятельных предметов экспозиции, что получило заслуженно высокую оценку посетителей. Как ни удивительно, большой популярностью пользовалась игровая программа фирмы «НИКИТА» «Защита Белого дома». По непроверенным слухам, циркулирующим на выставке, в настоящее время фирма работает над продолжением — «Штурм Белого дома». Но поживем — увидим.

Наряду с отечественными разработками начинают на наш рынок проникать и зарубежные игры. Интересно отметить, что речь идет о лицензионно чистой поставке компьютерных игр таких известных фирм, как, например, Silmarils. Так, при посредничестве фирмы «СОВИН» на выставке было представлено десять игр выпуска 1991-1993 годов по вполне приемлемым ценам (21-32 $). И, как показал ход выставки, первая попытка оказалась очень удачной.

Приятно отметить, что после двух лет нарастания сепаратизма и раздела имущества на российский рынок начинают возвращаться предприятия из стран СНГ. Примером может служить киевская фирма «Диалектика», предлагающая широкий спектр литературы по программированию. При этом киевляне не ограничились переводом технической документации (Object Windows для C++, Turbo Vision, FoxPro 2.0), но и позаботились о выпуске оригинальных авторских книг. Примером может служить книга А. Федорова и Д. Рогаткина «Borland Pascal в среде Windows», выпущенная тиражом 30000 экземпляров. Книга не предназначена для замены документации по системе, но может служить в качестве прекрасного справочного пособия, особенно при начальном знакомстве с системой Borland Pascal.

Надо признать, что украинские программисты, знакомые Вам еще по «Софт-панораме», наконец-то осознали потребность в собственных средствах массовой информации. И на стенде «Диалектики» посетителям был представлен первый номер киевского журнала для программистов «Софт-ревю». Первый блин оказался очень удачным, что можно было увидеть не только по счастливым лицам представителей фирмы и главного редактора г-на Штонды, но и по количеству желающих приобрести новый журнал. Наряду со статьями обзорного характера новое издание содержит полезную информацию и для программистов-практиков. В частности, первый номер «Софт-Ревю» содержит реконструированный исходный текст интегрированной среды Turbo Pascal 6.0, который представляет несомненный интерес для поклонников Turbo Vision.

Выставка произвела исключительно благоприятное впечатление. Нам как программистам было очень приятно увидеть, что наряду с прикладным программным обеспечением огромный интерес у посетителей вызывают и инструментальные системы. При этом речь идет не о традиционных программных продуктах фирм Borland и Microsoft, а о наших отечественных разработках. Так, знакомая Вам по нашим публикациям ЛПС Shift из Тамбова представила новые модули для объектно-ориентированной библиотеки Data Vision (*DBF Interface* и *Генератор Отчетов*). А украинское предприятие «Монолог» из г. Сумы продемонстрировало CASE-систему, предназначенную для создания информационных систем средней сложности.

Все больше появляется геоинформационных систем. И система ARCInfo больше не одинока. На этот бурно развивающийся в нашей стране сектор рынка

вступило ростовское АО «Информационные системы», предлагающее комплект информационных средств для создания геоинформационных систем. Большое впечатление производит также программное обеспечение фирмы «Трисофт», представившей на суд компьютерной общественности геоинформационную технологию СИНТЕКС/ТРИ, обеспечивающую ввод, визуализацию и комплексный анализ картографической информации.

Как оказалось, живы еще и разработчики аппаратных средств. Так, НИИСЧЕТМАШ представил на суд общественности плату графического адаптера, построенную на основе сигнального процессора TMS34010 и поддерживающую разрешение 1024*768 при 256 цветах. Производительность, безусловно, впечатляет, хотя стоимость (1000 $) несколько охлаждает пыл покупателей. Но, возможно, при серийном производстве все это удовольствие будет обходиться гораздо дешевле.

Безусловно, краткий обзор не позволяет исчерпывающе описать все достойные упоминания вещи. Можно было бы подробнее остановиться и на эргономичной «мыши», и на играх объемом около 2 Гигабайтов, и на многих других вещах. Но все это Вы сможете увидеть на стендах этих же фирм на любой другой выставке. Поэтому мы хотели бы написать о том, что можно встретить только на «SofTool».

Организаторы выставки не ограничились предоставлением выставочной площади и предоставлением традиционного для выставочного сервиса комплекса услуг. Осознавая, что «талантами Россия не оскудела», организовали конкурсные соревнования программистов, получившие названия «Первые SofTool'ийские Игры».

Цель состязаний состояла в том, чтобы дать возможность талантливым программистам проявить себя, объективно оценить свой профессиональный уровень и дать возможность фирмам привлечь наиболее грамотных специалистов. Организаторами соревнований выступил оргкомитет выставки и АО «ИНТЕХЦЕНТР».

Соревнования проводились в двух категориях: блиц-турнир и долгоборье.

Как формировались задания. Оргкомитет Игр обратился к ряду фирм с предложением сформулировать задания для конкурса. В результате был получен набор задач:

«ИНТЕХЦЕНТР» (совместно с European Computer Group) — оригинальные способы «озвучивания» компьютерной информации;

«COGNITIVE TECHNOLOGIES» — знание системы оптического распознавания TIGER;

фирма «КОРОНА» представила два класса задач: Novell expert (знание программных продуктов фирмы Novell) и HotLine Expert (знание программных продуктов фирм Borland, Microsoft, Symantec, Lotus);

«КУДИЦ» — конкурсы для школьников и студентов;

«ЛАН Крипто» — криптографические головоломки;

«МЕРКОР» — получение доступа к защищенной информации;

«МИКРОИНФОРМ» — знание программных продуктов «Мастер» и «Лексикон»;

«Мир ПК» и «СофтМаркет» — знание ассемблера;

«ФИЗТЕХСОФТ» — знание DOS и Command Processor;

«UNITECH» — программирование для сетей Novell Netware.

Задачи в классе «долгоборье» были заранее опубликованы в газетах «Человек и компьютер», «СофтМаркет», «Computer World», а блиц-турниры проводились непосредственно на специально выделенном стенде с 5 по 7 октября.

Общее количество программистов, участвоваших в Первых SofTool'ийских Играх в разных типах состязаний, составило около тысячи человек, в блиц-турнирах приняли участие более семидесяти. На торжественной церемонии закрытия Игр, состоявшейся 8 октября, Оргкомитет совместно с представителями фирм наградил 26 победителей и призеров. Среди призов аппаратное и лицензионное программное обеспечение, именные стипендии, книги и журналы по программированию, сувениры.

АО «ИНТЕХЦЕНТР» и дирекция выставки «SofTool» полагают, что такая деятельность поможет решению проблемы невостребованности творческого и профессионального потенциала России. Редакция журнала полностью поддерживает высокую оценку результатов Игр, данную ее организаторами, и планирует в следующем году также принять участие во Вторых SofTool'ийских Играх.

Условия участия будут публиковаться в компьютерной прессе, в том числе и в нашем журнале, начиная с апреля 1994 года. Блиц-турниры и подведение итогов в классе «долгоборье» состоятся непосредственно на выставке «SofTool-94». Там же состоится торжественная церемония закрытия Вторых SofTool'ийских Игр 1994 года с вручением дипломов и призов.

Мы ждем Вас на «SofTool-94»! ■

Монитор-Аспект 1994

Подписной купон на 1–е полугодие

1 | 2

Монитор Аспект

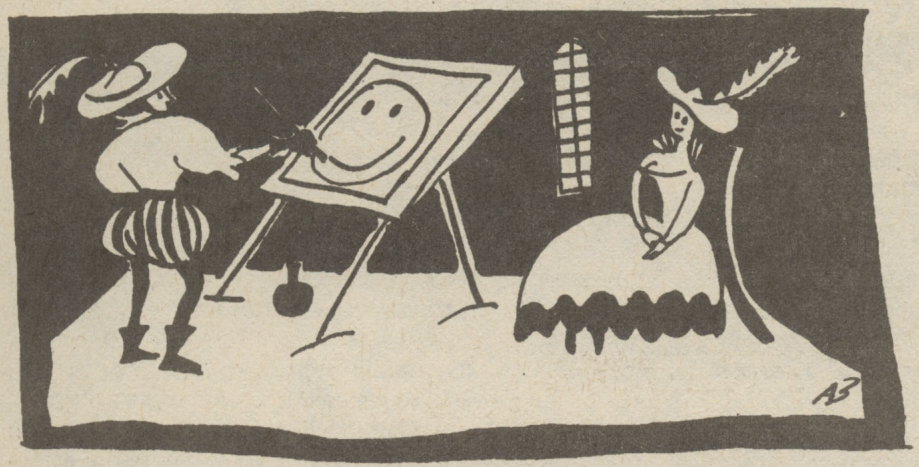
Адрес _____

Ф.И.О. или название организации _____

Стоимость подписки на 1 номер журнала "Монитор-Аспект" для России — 1200 рублей, для стран СНГ — 1800 рублей. В 1994 году будет выпущено 4 номера (подписка на полгода — соответственно 2400 руб. и 3600 руб.). Отметьте крестиками те номера журнала, которые Вы хотите получить. Перечислите сумму оплаты (действительна до 31 декабря 1993 г.) на счет НТО "Софт-Москва": р/с 467996 в Независимом Банке России, к/с 1161385 в РКЦ-2 ГУ ЦБ РФ по г. Москве, МФО 201779.
Вышлите подписной купон и квитанцию об оплате по адресу: 117419 Москва, а/я 765.

Язык программирования «Эллочка»

М. Трофимов



Людоедке Эллочке посвящается...

«Двенадцать программ с дублями и эпиграфами, или три триады программирования, а точнее третий лишний, а второй неправильный» — уф! Вот под таким не простым, а очень непростым названием предложил В. Очков «Этюды для программистов» в четвертом номере «Монитора» [1]. Если попробовать точнее определить жанр этой статьи, то я бы сказал, не этюды, а водевиль для программиста. Прежде всего дело, конечно, в водевильном названии, чрезвычайно удачно и емко отразившем суть статьи: пока не пригляделся, все кажется правильным, а вглядишься — видишь изъяны; как трещины в бокале — на вид целый, но к употреблению непригоден.

Действительно — «на вид целый», или, как принято писать в рецензиях, в статье много правильного. Так, например, хочется сразу отметить некоторые пассажи, пленяющие своей бесспорностью:

«Никто не будет спорить, что минус 29, а тем более минус 110 меньше, чем минус 2. Да, жаль, что у Дирака не было компьютера» [1].

А теперь про изъяны. Но сначала принесу извинения за вычурность — разбираясь в стилизациях [1], я тоже сбился на поэтический язык. Однако аналогии аналогиями, ирония иронией, но мое сравнение статьи с бокалом (равно как Дейкстры с Богом [1]) еще не есть доказательство. Поэтому, высказав свою точку зрения, я далее собираюсь ее обосновать. Но прежде было бы неплохо выяснить, о чем, собственно, статья Очкова, тем более, что, вопреки своей практике, «Монитор» на этот раз не дал резюме в оглавлении: посмотрите содержание четвертого номера — у всех авторских статей такие резюме есть, а у этой статьи нет. Анонс же из предыдущего номера выглядит интригующе-водевильно:

«Двенадцать примеров, разобрав которые, Вы убедитесь, что только начали свое знакомство с искусством программирования» [3].

Такое утверждение обязывает, по меньшей мере, к тому, чтобы его доказать. Как же проводятся доказательства по В. Очкову? В начале статьи приводятся три листинга с пояснениями, а далее следует фраза:

«Программы, приведенные в листингах 2 и 3, *доказывают* [курсив мой — М.Т.], что при особом желании можно обойтись без альтернативы: в программах листинга 2 она есть, а в программах листинга 3 — нет».

Но, простите, что может «доказать» программа? (если, конечно, она не относится к ограниченному классу узкоспециальных программ искусственного интеллекта по доказательству теорем). Доказывает, как известно, автор, а не программа, и частных случаев нескольких тривиальных программ недостаточно, чтобы обосновать общие выводы типа: «Основная структурная теорема в течение почти 30 лет давалась в неверной формулировке. Правильная редакция такова...» [1]

Из курса математики для средней школы каждому, кто там учился, должно быть хорошо известно, что для доказательства, например, формулы суммы членов арифметической прогрессии недостаточно рассмотреть какие-нибудь две конкретные последовательности, — в школе учат вести доказательство по индукции; или под словом «доказать» автор понимает совсем не то? Тогда это своеобразный авторский жаргон, могущий не прояснить смысл, но запутать читателя... (Кстати, на подходе к финалу и сам автор любезно интересуется, не запутался ли читатель). И тут у меня возникло сомнение, а не делается ли попытка сознательно запутать читателя, применяя для этого не только жаргон, но и другие, более чем своеобразные, формы построения статьи? К примеру, все эти слова про «триады-диады»: «Магия цифры три не обошла стороной и автора, который в статье с тремя названиями поместил 12 (четырежды три) программ, ссылается на 9 (трижды три) литературных источни-

ков, упоминает шесть (дважды три) языков программирования...» [1]

Что это, пародия на структурированный текст, насмешка над структурным программированием или желание создать как можно больше маскировочного шума? А между тем подобные ухищрения ни к чему (как уже отмечалось, в статье много правильного), и автор при желании действительно мог бы аргументировать свою позицию, рассматривая условный оператор (альтернативу) как частный случай цикла и показав избыточность универсальных языков программирования, таких как Pascal, BASIC и C. Все это можно сделать, воспользовавшись работами того же Дейкстры и его последователей. При том что, если, конечно, я правильно понял одну из целей статьи, — она заключалась в демонстрации меньшей избыточности BASIC по сравнению с Pascal и C — здесь так и хочется продолжить мысль [1]: «жаль, что у Дирака не было компьютера», — и он (Дирак) не знал BASIC. Жаль также, что автор повторил фразу про отсутствие PC у Дирака всего лишь два раза — такая бы триада получилась! Увы! Как начнешь иронизировать, ох, как трудно остановиться! Но на иронию принято отвечать иронией, а если серьезно, много трудов не потребуется — достаточно посмотреть в учебник (ух, как жаль, что у Дирака не было учебников по языку Pascal, увы, и сейчас они есть, видимо, далеко не у всех!): так, избыточность отмечается во многих учебниках по программированию. Заглянем хотя бы в книгу [4] (увы, не вошедшую в «трижды три» литературных источников [1]): «Вообще говоря, оператор перехода вместе с условным оператором является наиболее универсальным средством управления порядком выполнения других основных операторов. В частности, любой циклический процесс, задаваемый каким-либо оператором цикла, может быть задан без использования операторов цикла».

языки

Борясь с избыточностью языка и упрощая сверх меры, мы оказываемся в проигрыше.

Например, берем за основу определение оператора цикла с предусловием

```
while cond do something
```

Здесь cond — условие, something — оператор или последовательность операторов.

Можно показать, что данный цикл будет эквивалентен последовательности операторов

```
1: if not (cond) then goto 2;
  something; goto 1;
2:
```

Доказав это утверждение и подобные для других операторов цикла — цикла с постусловием и цикла с параметром, мы можем назвать операторы if и goto главными и обойтись без операторов цикла (можно предложить и другой «базис», как сделал это Очков — суть не изменится!). Более того, в целях борьбы с избыточностью вообще можно обойтись одним «обобщенным оператором», рассматривая оператор безусловного перехода как частный случай условного, то есть каждый оператор (op), не требующий ветвления, заменим, например, такой последовательностью

```
m1: if true then begin op; goto m2 end;
m2:
```

При этом на долю op достанется только оператор присваивания, поскольку без процедур, функций и их вызовов также можно обойтись. Можно обойтись, кстати, и без ключевых слов: достаточно указывать только содержательные части обобщенного оператора. Например, в следующем формате:

```
<метка><условие><переменная>
<выражение><метка перехода>
```

в результате получим что-то вроде

```
110 x<y a 10*c 112
```

Это будет эквивалентно

```
110: if x<y then begin a:= 10*c; goto 112 end;
```

Можно пойти и дальше: с операторами разобрались, теперь подсократим избыточность выражений. Зачем, в самом деле, в Pascal зарезервированы and и or? Достаточно только одного, например, and, пользуясь им, а также отрицанием not, мы всегда сможем обойтись без or, поскольку

```
to_be or not_to_be = not (not to_be and
                          not not_to_be)
```

Или зачем нужен знак «не равно» (<>), когда x<>y эквивалентно not (x=y)?

Далее, ввод и вывод может быть сведен к обращениям к зарезервированной буферной переменной con — консоль. Например,

```
218 true con 'hello' 219
```

по определению будет эквивалентно

```
write ('hello')
```

И

```
219 true x con 220
```

будет определено как

```
read (x)
```

Что касается некоторых стандартных функций, таких как sin или odd, то их, пожалуй, можно оставить, минимизировав при этом их количество:

```
220 odd (i) y sin (x) 221
```

В итоге получим сверхэкономный язык, основанный на сверхлаконичном словаре — этакий словарь людоедки Эллочки для программистов. Вся беда только в том, что, как известно, скупой платит дважды, или, по эпиграфу из [1], «жадность фраера погубит», — многие программы на таком неумеренно лаконичном языке окажутся многословнее своих неэкономных собратьев. И действительно, Turbo Pascal фрагмент

```
while (a<>0) or (b<>0) do
begin
  dec (a); dec (b)
end;
```

будет содержать гораздо меньше слов (символов языка), чем аналогичный на языке «Эллочка»

```
1 not (not (a=0) and not (b=0)) a a 4
2 true a a-1 3
3 true b b-1 1
4
```

(Здесь в строке 1 для заполнения поля оператора ор в качестве пустого оператора, не определенного в «Эллочке», применена конструкция, эквивалентная в Pascal a:= a.)

Несколько тоньше вопросы об избыточности и неполноте системы циклов языков Pascal и C (ограничимся здесь одним — Pascal). Избыточность: да, конечно, можно ограничиться циклом одного вида и, например, вместо

```
for i:=1 to 10 do write (i);
```

писать больший текст

```
i:=1;
while i<=10 do
  begin
     write (i);
     i:=i+1
  end;
```

Только вот зачем, непонятно. И где же тут искусство, обещанное в анонсе [3]?

Неполнота:

«На языках C и Pascal цикл с выходом из середины можно получить только через его «запихивание» в прокрустово ложе цикла for..., либо цикла while..., либо цикла do (repeat...until...)» [1].

Ответим на это другой цитатой — эпиграфом из [1]: «Искусство — это чувство меры». Борясь с избыточностью, упрощая сверх меры, как в языке «Эллочка», мы оказываемся в проигрыше — чрезмерное усложнение также до добра не доведет, и процитированный в статье Очкова отрывок [5] — подходящая тому иллюстрация, хотя, конечно, при желании в сборнике [5] можно найти и более удачные цитаты из других статей. Со своей стороны добавим к списку эпиграфов, использованных в [1], еще одну общеизвестную мудрость — «за все приходится платить» — в том числе и за удобную структуру программы.

Возможно, кому-то кажется удобным, не задумываясь, множить точки выхода, точно так же кому-то удобнее беспорядочно разбрасывать по тексту программы метки и операторы перехода к ним, приводя программу к виду, определяемому термином «спагетти». Структурный же подход требует от программиста дисциплины, несколько ограничивая его произвол (что почти всегда во благо), но в отдельных случаях доставляя и неудобства. Впрочем, так ли уже часто выход из середины цикла бывает необходимым? — По моим наблюдениям, не очень, но уж если он действительно нужен, то, думаю, не надо расстраиваться: один-два оператора перехода и метка стилю вашей программы повредить никак не сможет. Полагаю поэтому, что отмеченные в [1] недостатки циклов Pascal и C — более теоретического, нежели практического характера. Ну что же, в таком случае еще одна мудрость — «увы! идеал недостижим».

И на этом можно было бы поставить точку, но както странно: что же, только в этой избитой фразе и заключается смысл статьи Очкова — показать, что Pascal и C — неидеальные языки? Разумеется, и все другие упомянутые в статье шесть языков тоже не являются идеальными. Во всяком случае, в выводах [1] какие бы то ни было оценки (типа «BASIC лучше») отсутствуют. Правда, по ходу дела в самой статье делаются намеки на такой вывод, и кончается она вопросом «не пришла ли пора реабилитации входов в цикл, минуя заголовок», чтобы повысить гибкость языка программирования. Задам встречный риторический вопрос: конструкция «спагетти» самая гибкая, но где же чувство меры? И давайте ограничимся этим вопросом, сославшись на горы бумаги, исписанной в спорах о структурном программировании и победе «структурной революции». Кстати, тема «оператор goto и управляющие структуры» неисчерпаема — для примера сошлюсь на сравнительно недавние работы [2], [6], оригинальные по содержанию (несмотря на избитость темы), правда, не столь оригинальные по форме, как [1].

Итак, в статье Очкова, написанной в крайне своеобразной манере и крайне своеобразным языком, языкам Pascal и C делаются два прямо противоположных упрека: в избыточности и недостаточности управляющих конструкций. Аргументация построена на осно-

вании анализа 12 коротких частных и достаточно тривиальных примеров (кстати, еще один вопрос автору: а нельзя ли заменить триады примеров диадами, а может, и монадами? — Может, я чего-то не понял, но мне показалось, что однотипных примеров в избытке. И если это так, то почему их «четырежды три», а не трижды три по три?). Вот, собственно, и все. Получилось негусто, боюсь, потому, что многие важные моменты просто утонули в слишком пышных формах.

И очень жаль, ведь, будь статья написана проще, и займись автор более серьезной аргументацией вместо игр «Три раза по три», читалось бы это с большим интересом как сторонниками, так и противниками структурного подхода в программировании. Хочется надеяться, что не все потеряно и В. Очков на страницах «Монитора» даст по-настоящему аргументированные ответы на мои недоуменные вопросы (в частности: о чем статья? что может доказать программа? чем автору досадила избыточность языков программирования? — и не только программирования, ведь естественные языки тоже избыточны и так далее). Надеюсь также, что будет выполнено и обещание, данное в постскриптуме [1], и мы узнаем, чем грозит болезнь объектно-ориентированного программирования. В этом обещании уже предваряется тема «Все ли там (в ООП — М.Т.) логично, и нельзя ли обойтись диадой (вместо триады — М.Т.)».

Действительно, интересно. Хочется только заранее попросить: давайте перестанем путать избыточность и логичность. Например, «триада»

```
a = 5
b = 5
a = b
```

избыточна, и без потери ценной информации можно обойтись «диадой», но и данная триада, и сделанные из нее диады типа

```
a = 5
a = b
```

или

```
a = 5
b = 5
```

будут воистину логичны. А вот «триада»

```
a = 5
b = 6
a = b
```

будет неизбыточна и нелогична. Третья «триада»

```
a = 5
b = 6
c = 7
```

логична и неизбыточна. Или мы по-разному понимаем эти термины? Так давайте сначала договоримся о терминологии!

К списку названных терминов, значение которых осталось для меня неясным в статье [1], я бы добавил еще и термин «алгоритм»: «...альтернатива — это средство ускоренного «путешествия» по алгоритму (а не по программе? — М.Т.) только в одну сторону — сверху вниз и слева направо (а если алгоритм записан в словесной форме на иврите, то справа налево? — М.Т.), если смотреть на листинг (листинг чего — алгоритма или программы — М.Т.), а цикл — в обе».

Кстати, есть такое понятие, как степень детализации алгоритма, но я никогда не встречал понятия «детализация программы или исходного/исполняемого кода» — код всегда детален, то есть пригоден для однозначного исполнения на компьютере. Но это уже тема отдельной дискуссии.

Темой отдельного обсуждения является и другая недавняя статья г-на Очкова [7], содержащая текстуально совпадающие с [1] фрагменты. ■

Литература

1. В. Очков. Двенадцать программ с дублями и эпиграфами, или три триады программирования, а точнее третий лишний, а второй неправильный// Монитор.– 1993.– № 4.
2. D. Jonsson. Next: The Elimination of Goto-Patches?// ACM SIGPLAN Notices.– 1989.– Vol. 24.– № 3.
3. «Монитор» — версия 4.93. Анонс// Монитор.– 1993.– № 3.
4. В.Г. Абрамов, Н.П. Трифонов, Г.Н. Трифонова. Введение в язык Паскаль. — М.: Наука, 1988.
5. Языки программирования Ада, Си, Паскаль. Сравнение и оценка. — М.: Радио и связь, 1989.
6. T. Cooper. Goto? Yes, goto!// Computer Language.– May 1991.
7. В. Очков. Turbo Pascal 7.0. Взгляд со стороны// КомпьютерПресс.– 1993.– № 7.

НТО

Софт·Москва

предлагает широкий спектр редакционно-издательских и полиграфических работ:

- литературное редактирование
- разработку рекламы для печатной продукции
- изготовление:
  комплектов документации для программных продуктов;
  бланков, листовок, проспектов;
  сертификатов ценных бумаг;
  визиток, наклеек, этикеток, приглашений;
  слайдов;
  изданий за счет авторов.

Наш адрес: 117419 Москва, а/я 765.

Тел.(095) 943-25-53, 238-99-33. Факс (095) 238-32-78

Профессиональная работа в среде Microsoft BASIC

А. Колесов

Часть 3. Когда программы становятся большими

(Начало см. в № 6.93)

Примерно год назад, работая с программами, использующими режим обработки программных прерываний, мы обратили внимание, что иногда они начинали отрабатывать режим HELP не так, как нам бы этого хотелось. Например, при нажатии F1 в момент нахождения в меню нужная подсказка появлялась не сразу, а только после выхода из меню, когда никакой подсказки нам уже не требуется. Причем это происходило при работе как в среде QB.EXE, так и исполняемого модуля.

Анализ подобной ситуации потребовал от нас детального изучения организации режима программных прерываний, но время, потраченное на это, было вполне оправдано. Мы не только добились правильной работы программы — нам удалось сократить размеры наших EXE модулей на 30%!

/V и /W — включение средств программной обработки событий

С помощью ключей /V, /W в объектный модуль вводятся вспомогательные средства обработки прерываний событий, связанных с управлением функциональными клавишами (KEY), последовательными портами (COM), джойстиком (STRIG), световым пером (PEN), таймером (TIMER), музыкой (PLAY) и произвольным процессом, определяемым пользователем (UEVENT). Конструкция UEVENT впервые появилась в версии 4.5, и надо иметь в виду, что для ее использования в среде QB.EXE требуется загрузка любой Quick библиотеки, например, QB.QLB.

Эти режимы реализуются с помощью операторов типа «ON event», «event {ON|OFF|STOP}», KEY (). В дальнейшем, имея в виду эту группу операторов, мы будем использовать термин «ON event», и поскольку чаще всего в программах используются прерывания от функциональных клавиш, в дальнейшем общая проблема обработки событий будет рассматриваться именно на их примере.

В описании компилятора говорится, что ключ /V обеспечивает проверку наступления инициализированных событий между каждым оператором (STATEMENT), а /W — между каждой строкой (LINE) текста программы. Такое определение требует пояснения механизма обработки этого вида прерываний.

Смысл использования данной конструкции заключается в том, чтобы при наступлении определенного события, например, нажатии функциональной клавиши, прервать выполнение программы, зафиксировать ее текущее состояние, передать управление некоторой процедуре обработки этого события и после завершения обработки восстановить состояние и продолжить выполнение прерванной программы.

Рассматривая проблему отслеживания наступления событий, необходимо отметить, что возможны два основных способа реализации контроля: в режиме опроса и в режиме прерывания.

В режиме опроса непосредственно в текст программы включаются операторы анализа наступления того или иного события, например:

```
IF TestEvent THEN {CALL | GOSUB | ...}
```

Естественно, программист сам определяет места программы, где он считает необходимым проводить опрос событий, последовательность опроса, решает проблему сохранения/восстановления состояния программы и прочее.

Режим прерывания подразумевает такую логику работы. Сначала выполняется некоторая процедура инициализации контролируемого внешнего процесса, а затем начинает выполняться программа, которая в общем случае и понятия не имеет о контроле каких-то событий и о том, что ее выполнение могут прервать, продолжить и так далее.

Классическим примером такой логики работы являются операционные системы, обеспечивающие работу программ в реальном масштабе времени. При этом они решают проблемы приоритета прерываний, организации очередей запросов и тому подобные. При работе в MS-DOS мы сталкивается с алгоритмами программных прерываний при использовании каких-то резидентных утилит, например, SIDEKICK калькулятора.

Говоря о контроле событий в режимах опроса и прерывания, можно привести такую аналогию. Чтобы узнать о появлении некоторого товара в торговой сети, можно периодически или постоянно обзванивать все магазины, а можно послать в эти магазины запросы и ждать звонка из магазина. Очевидно, что выбор алгоритма контроля зависит от характера решаемой задачи, срочности, увязки необходимости информации с выполнением других дел и прочего.

языки

Аккуратное использование средств программной обработки событий может сократить размеры EXE модулей на 30%.

Возвращаясь теперь к ключам /V и /W, следует четко представлять, что BASIC с помощью конструкций «ON event» реализует некий режим псевдопрерываний. На самом деле операторы «ON event» и «event ON» действительно проводят установку внутренних флагов контролируемых событий в специальной внутренней процедуре BASIC, но отслеживание событий происходит с помощью программного опроса. При наличии ключей /V, /W компилятор просто автоматически расставляет запрос к внутренней процедуре B$EVCK перед каждым оператором или строкой (звоним в магазины каждые пять минут!). Этим и вызвано увеличение объема объектных модулей программы [2] — при классической схеме механизма прерываний размер программы никак не зависит от процедур контроля за событиями.

Еще одно замечание относительно отличия ключей /V и /W. Очевидно, что строка состоит из одного или нескольких операторов, поэтому комбинация /V/W равносильна /V. Но что такое строка программы, перед которой производится проверка по ключу /W? Здесь надо быть внимательным: в данном случае под термином LINE подразумеваются не строки ASCII файла исходного модуля, а последовательность операторов, которая помечена цифровой или символьной меткой. Поэтому если в тексте Вашей программы не будет ни одной метки, то в случае использования только /W в нее не будут включены средства проверки событий.

Что делать дальше?

Наличие в исходном программном модуле хотя бы одного оператора типа «ON event» требует обязательного использования ключей /V или /W при его компиляции. Но в отличие от случая /E/X [2] обработка программных прерываний возможна в любом модуле, в который включены соответствующие средства их контроля. Поэтому система QB.EXE делает очень просто: она компилирует ВСЕ исходные модули программы с ключами /V/W. Результатом этого является то, что, если Вы включили в свою программу хотя бы один оператор типа ON event, Ваш EXE модуль автоматически увеличивается на 20-30%.

Вы хотите сэкономить эту память? Тогда читайте дальше. Решением этой проблемы являются следующие положения:

1. Если у Вас нет особой нужды использовать конструкции «ON event», лучше обойдитесь без них. Опыт нашего анализа прикладных программ различных авторов показывает, что в 95% случаев ее применение связано с контролем нажатия клавиш управления (функциональных, клавиш управления курсором и прочих), которое может быть значительно проще реализовано с помощью оператора INKEY$ в режиме опроса.
2. Если Вам все-таки нужно применить именно режим обработки программных прерываний, то, используя довольно простые приемы программирования, можно свести увеличение размеров исполняемого модуля к минимуму (часто практически к нулю). Далее обсуждается эта проблема.

Ее принципиальное решение заключается в том, что для реализации программных прерываний нужно разместить средства опроса контролируемых событий не по всей программе, а только в тех ее фрагментах, в которых это действительно необходимо. Для этого при разработке программы необходимо сразу выделять в отдельные модули (исходные файлы) все процедуры инициализации обработки программных прерываний и те процедуры, где Вы будете производить контроль событий. Затем при создании EXE модуля необходимо откомпилировать с ключами /V/W только эти исходные модули, а остальные модули программы — без этих ключей. Соответственно увеличение исполняемого модуля будет пропорционально соотношению этих двух групп модулей.

Пример конкретной реализации такого подхода

Характерным примером использования программного прерывания является реализация некоторого режима HELP, когда при нажатии какой-то клавиши, например, F1, нужно прервать текущее выполнение программы и передать на время управление процедуре вывода на экран соответствующего сообщения, а затем продолжить выполнение прерванного фрагмента.

В своих программах мы довольно часто используем в качестве HELP просмотр фрагмента ранее подготовленного текстового файла. Для реализации такой конструкции нужно прежде всего создать отдельный модуль инициализации контроля этой клавиши. В нашем случае мы используем готовый вспомогательный модуль — HELP_ON.BAS (листинг 1), включающий небольшие процедуры HelpFNon и HelpFNoff — инициализации и сброса контроля события (нажатия функциональной клавиши) — и подпрограмму UsgtF1, которая, принимая управление при наступлении события, сразу передает управление внешней процедуре его обработки — HelpText.

Проблему выделения фрагментов программы, в которых ведется отслеживание контролируемых событий, мы решили следующим образом. Поскольку мы делаем диалоговые программы, то основное время их работы приходится на ожидание нажатия клавиш оператором. Это ввод данных, выбор позиций меню и прочее. Здесь следует заметить, что, как и многие другие программисты, при создании оконных интерфейсов для ввода данных мы не работаем с оператором INPUT, а используем собственную процедуру KzInputString, которая представляет собой небольшой редактор символьной строки. При этом ввод кодов нажатых клавиш осуществляется через процедуру InkeyWaite (модуль INKEYW.BAS, листинг 2).

Кроме всего прочего, такая реализация ввода данных обеспечивает возможность прерывания программы и вызова HELP в любой момент ввода данных, а при использовании INPUT прерывание может произойти только после завершения ввода. И еще такой подход позволил нам очень просто решить проблемы ввода русской буквы «р» в QuickBASIC — в этом случае мы используем другой вариант процедуры InkeyWaite (модуль INKEYW45.BAS, листинг 2), в котором ввод кодов клавиш выполняется не оператором INKEY$, а через системное прерывание.

Совершенно естественно, что во всех вспомогательных процедурах инструментальных библиотек оконного интерфейса все операции ожидания нажатия клавиш выполняются через процедуру InkeyWaite. Таким образом, во всех диалоговых программах контроль наступления каких-то событий может быть полностью локализован в трех операторах этой процедуры!

Таким образом, для реализации программной обработки прерываний достаточно откомпилировать с ключами /V/W только небольшие модули HELP_ON.BAS, INKEYW.BAS (рис. 1). Этот подход к

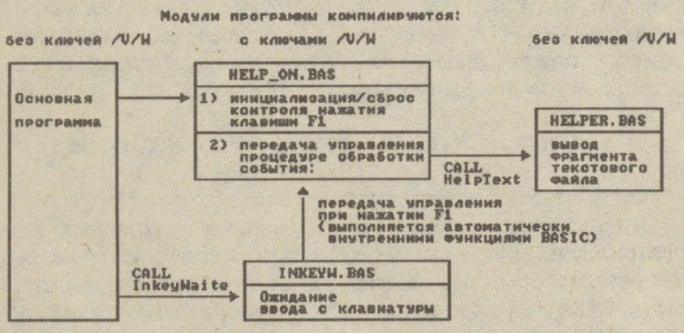


Рис. 1. Пример структуры программы при реализации обработки контролируемых событий в режиме прерывания

использованию конструкций «ON event» достаточно просто реализуется и в других случаях. Например, при создании расчетных программ можно сделать в некоторых контрольных точках обращения к фиктивной процедуре типа CALL TestEvent:

```
SUB TestEvent
END SUB
```

Процедуру надо поместить в отдельный модуль и откомпилировать с ключами /V/W.

Управление режимами компиляции

Теперь надо решить вопрос, как реализовать индивидуальное управление режимами компиляции отдельных модулей. Конечно, можно проводить всю процедуру создания EXE модуля вне среды QB.EXE, запуская BC.EXE и LINK.EXE командной строкой, но это будет очень неудобно. В принципе, достаточно просто написать свою автономную программу, которая будет выполнять функцию «MAKE EXE» с учетом любых пожеланий пользователя. Хотя и здесь есть некоторые проблемы, так как нам придется самим определять некоторые ключи компиляции, которые QB.EXE устанавливает сам, исходя из анализа исходного текста каждого модуля.

Поэтому мы решили сделать по-другому: работая в среде QB.EXE, при компиляции мы используем небольшую вспомогательную программу BC_VW.EXE (листинг 3), которая производит необходимую коррекцию командной строки. То есть вместо последовательности «QB.EXE -> BC.EXE» мы реализуем «QB.EXE -> BC_VW.EXE -> BC.EXE» (рис. 2).

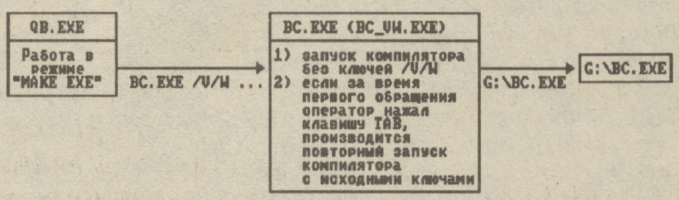


Рис. 2. Управление ключами компиляции с использованием программы BC_VW.EWE

Установка такого режима выполняется командным файлом BCVW_SET.BAT:

```
@echo off
@echo Установка режима "блокировка ключей /W/V"
rem скопировать компилятор на виртуальный диск:
copy bc.exe g:\bc.exe
rem переименовать программы:
rename bc.exe bc.old
rename bc_vw.exe bc.exe
```

А сброс — BCVW_NUL.BAT:

```
rename bc.exe bc_vw.exe
rename bc.old bc.exe
```

Установив режим BCVW_SET, при создании исполняемого модуля программа BC_VW.EXE перехватывает обращение к BC.EXE, удаляет из командной строки ключи /V/W и запускает компилятор BC.EXE без них. Если пользователь решит, что компилировать данный модуль нужно все-таки с этим ключами, ему достаточно нажать клавишу TAB — будет произведена повторная компиляция с исходной командной строкой. Эта процедура не приносит особых хлопот пользователю. Во-первых, такой режим работы используется не очень часто. Во-вторых, вмешательство программиста в управление компиляцией реально бывает нужно для двух-трех модулей, имена которых мы обычно записываем в начало MAK файла. Таким образом, после специальной обработки нескольких первых модулей программы пользователь может рас-

слабиться и спокойно ждать завершения создания EXE модуля.

Следует отметить, что при использовании виртуального диска (в нашем случае G:) для хранения компилятора BC.EXE время создания исполняемого модуля в режиме BCVW_SET даже немного сокращается, так как суммарное время загрузки небольшой программы BC_VW.EXE с жесткого диска и BC.EXE с виртуального меньше времени загрузки BC.EXE с жесткого диска. Разумеется, программу BC_VW.EXE надо создавать в режиме «Stand Alone»: так быстрее производятся операции загрузки и передачи управления между программами [2].

Работа с Quick библиотеками требует особого внимания

Некоторые проблемы с реализацией конструкций «ON event» могут возникнуть при использовании Quick библиотек.

Предположим, что, применяя ранее описанные модули HELP_ON.BAS и INKEYW.BAS, Вы захотите сделать Quick библиотеку для своей программы. Причем модуль INKEYW.BAS Вы поместите в библиотеку (без нее не будут работать многие вспомогательные процедуры — меню, ввод и прочие), а HELP_ON.BAS решите загружать в виде исходного модуля.

Создавая Quick библиотеку в среде QB.EXE, Вы загрузите все вспомогательные модули [2], и запустите операцию «MAKE Library». Но так как среди включенных в библиотеку модулей ни один не использует операторы типа «ON event», то QB.EXE просто не будет устанавливать ключи /V/W и средства контроля событий в библиотеку включены не будут!

Результатом этого явится неправильная логика работы программы, описанная в начале этой статьи. Дело в том, что процедура MENU была также включена в библиотеку. Поэтому при нажатии F1 обработка прерывания происходила только при выходе из процедур библиотеки в процедуры, которые были загружены в виде исходных модулей. Аналогично при работе с Quick библиотеками в среде QB.EXE происходит отработка аварийного прерывания программ по CTRL-BREAK.

Решение описанной проблемы является довольно простым: надо отредактировать полученную в среде QB.EXE библиотеку (например, LIB0) следующим образом:

1) откомпилировать модуль INKEYW.BAS, включив в него средства контроля событий:

```
BC.EXE INKEYW.BAS /V/W
```

2) создать новую библиотеку LIB1, заменить в библиотеке LIB0 модуль INKEYW:

```
объектная:
        LIB.EXE LIB0.LIB-INKEYW+INKEYW,,LIB1.LIB;
Quick:
        LINK.EXE /Q LIB1.LIB,LIB1.QLB,,BQLB45.LIB;
```

При работе с библиотеками следует иметь в виду еще одно важное обстоятельство. Если Вы будете использовать библиотеку LIB1.QLB, то при создании исполняемого модуля среда QB.EXE автоматически установит режим компиляции с ключами /V/W для всех исходных модулей программы, даже если среди них нет ни одного с оператором типа «ON event». Поэтому при работе с некоторыми библиотеками стандартной конфигурации мы сразу создаем два варианта, например, упомянутые ранее LIB0 и LIB1, которые применяем соответственно при разработке программ, не использующих и использующих режим обработки программных прерываний.

Оптимизация процедуры создания исполняемого модуля

Команда «MAKE EXE» в среде QB.EXE выполняется следующим образом: производится последовательная компиляция ВСЕХ исходных модулей программы, а затем полученные объектные модули компонуются в исполняемую программу. При этом известно, что проблема оптимизации процедуры создания EXE модуля в среде QuickBASIC в общем-то никого особенно не волновала — ни разработчиков языка, ни его пользователей.

языки

Оптимизация процедуры создания EXE модуля — перекомпиляция только исправленных исходных файлов.

С одной стороны, общая логика разработки программы в QuickBASIC такова, что основной объем работы при отладке программы выполняется в среде интерпретатора, а создание исполняемого модуля производится только для готовой программы, то есть операция «MAKE EXE» является разовой и время ее выполнения не играет никакого значения. С другой стороны, многим пользователям QuickBASIC вообще впервые выпало счастье работы с EXE модулями и они были вполне довольны любой процедурой его создания.

Тем не менее, объем прикладных программ потихоньку увеличивался и количество исходных модулей в них росло. Например, средний размер EXE модулей наших программ составляет около 100 Кб (режим «Requiring BRUN45.EXE»). Они включают до 60-70 отдельных исходных модулей формата «TEXT» общим объемом 5000-8000 строк или 200-300 Кб. И соответственно время создания исполняемого модуля таких программы составляет на AT/286 уже не 20-30 сек, а 4-5 минут. При этом надо еще раз подчеркнуть, что разработка таких программ в приемлемые исторические сроки (мы делаем программные комплексы, включающие до десятка и более EXE модулей) невозможна без эффективного использования готовых инструментальных средств [1], — на их долю приходится 60-80% исходного текста программы.

С другой стороны, саму операцию «MAKE EXE» приходится выполнять все чаще. Например, комплексную отладку пакета программ надо выполнять в режиме взаимодействия EXE модулей, и обнаружение локальной ошибки требует перекомпоновки программы. Кроме того, мы периодически сталкиваемся с ситуацией, когда ошибка проявляется только в исполняемом модуле программы (это является следст-

вием наличия в системе двух трансляторов — интерпретатора и компилятора [1]).

Короче говоря, вопрос оптимизации процедуры «MAKE EXE» назрел. Принципиальным решением проблемы является такая организация работы, когда повторной компиляции подвергаются только те исходные модули программы, которые были исправлены во время отладки. Это делается следующим образом: компилятор или некоторая управляющая программа сравнивает даты и время создания (коррекции) исходного и объектного модулей и выполняет компиляцию, только если объектного модуля просто не существует или время его создания меньше времени модификации исходного файла.

Такой режим есть практически во всех современных компилирующих системах. В принципе, проблема оптимизации создания новых файлов по описанному выше принципу может быть решена с помощью некоторых универсальных утилит, например, автономной программы Microsoft NMAKE, которая, в частности, входит в состав пакета BASIC PDS 7.1. Но все-таки практически пользоваться ею не очень удобно — нужно выходить из среды, создавать управляющий файл и так далее, не говоря уже о том, что права на ее использование нужно сначала купить, а затем научиться работать с ней. Нам кажется, что мы нашли более простое решение.

Как и в случае с управлением ключами компиляции /V/W, мы написали небольшую программу MYMAKE.EXE (листинг 4), которая производит сравнение времени создания исходного и объектного файлов и запускает компилятор в случае необходимости, работая в режиме препроцессора компилятора (рис. 3):

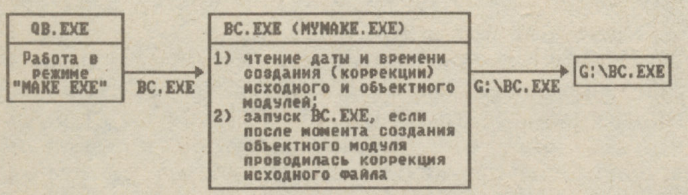


Рис. 3. Оптимизация процедуры создания нового исполняемого модуля программы

```
QB.EXE -> MYMAKE.EXE -> BC.EXE
```

С учетом размещения BC.EXE на виртуальном диске и формирования MYMAKE.EXE в режиме «Stand Alone» такая конструкция работает даже немного быстрее обычного режима компиляции и в случае создания нового объектного модуля.

Основная проблема быстрого чтения атрибутов файла решается с помощью процедур модуля CALLDIR.BAS (листинг 5) путем вызова функций DOS. Обращаем внимание, что среди них есть очень полезная процедура чтения списка каталога файлов — FormDirNew.

Установка режима MYMAKE производится следующим образом:

```
rem скопировать компилятор на виртуальный диск:
copy bc.exe g:\bc.exe
rem переименовать программы:
rename bc.exe bc.old
rename mymake.exe bc.exe
```

Восстановление исходного режима:

```
rename bc.exe mymake.exe
rename bc.old bc.exe
```

Поскольку мы периодически изменяем обычный режим компиляции в системе, устанавливая BC_VW или MYMAKE, то, чтобы не путаться с проблемой его восстановления, мы используем единый командный файл BC_RESET.BAT (листинг 6). Мы также его вставили в AUTOEXEC.BAT, чтобы автоматически производить установку исходного режима компиляции при загрузке системы.

Естественно, возможность оптимизации процедуры создания исполняемого модуля подразумевает соответствующий стиль программирования: программа должна состоять из отдельных относительно небольших исходных модулей. В этом случае исправление одиночной ошибки — наиболее частая ситуация при комплексной отладке и при опытной эксплуатации — приведет к необходимости коррекции и перекомпиляции только одного файла, а не 60-70, как в обычном варианте. И, разумеется, не следует удалять после компоновки полученные объектные модули, которые многие пользователи BASIC рассматривают как некоторый мусор.

Затронутые в статье вопросы, естественно, не претендуют на полное освещение проблем, связанных с созданием больших программ. Собственно, с маленькими программами вообще проблем никаких нет. Авторы цикла статей рассматривают эту публикацию как начало обмена опытом пользователей языка Microsoft BASIC и выражают свою готовность продолжить начатый разговор. ■

Литература

1. А. Колесов, О. Павлова. Профессиональная работа в среде Microsoft QuickBASIC. Ч. 1, 2// Монитор.– 1993.– № 2, 3.
2. А. Колесов. Профессиональная работа в среде Microsoft BASIC. Ч. 3. Когда программы становятся большими// Монитор.– 1993.– № 6.

Вниманию организаций и частных лиц!

Рекламное агентство компьютерного журнала «Монитор» приглашает к взаимному сотрудничеству по представительству интересов в городах России (вне Москвы), государствах СНГ и ближнего зарубежья (сбор рекламы и размещение ее в местной прессе, проведение рекламных кампаний и так далее).

С предложениями обращаться по телефонам:

отдел рекламы:	*(095) 903-46-26*
редакция:	*(095) 238-99-33*
факс:	*(095) 238-32-78*

Приобрести все номера журнала «Монитор» за 1992 и 1993 годы можно в магазине «Дискеты» по адресу:

Москва, Оружейный пер., д. 15

Телефон: 251-13-17

Листинг 1.
Модуль HELP_ON.BAS — процедуры инициализации контроля программного прерывания при нажатии функциональной клавиши

```
DECLARE SUB HelpStoreScreen0 (kod%)
DECLARE SUB HelpText (NameF$, Key$)
DEFINT I-N
'*********************************************
'**********  Kolesov QB Tools     ************
'*     БИБЛИОТЕКА подпрограмм QB_HELP         *
'*********************************************
'*           Модуль HELP_ON.BAS               *
'* Инициализация обращения по прерыванию      *
'*              и запуск HELP                 *
'*********************************************
' ВНЕШНИЕ ССЫЛКИ: HELPER.BAS, HELP_ST.BAS
'*********************************************
'* Состав процедур:
'  CALL HelpFNon(NameFile$,NameRazd$,
'  Nstr%,Npage%,kod%,Khelp%) -
'        инициализация обращения
'        по прерыванию - нажатию F...(Khelp%)
'  CALL HelpFNoff -
'        сброс обращения
'        по прерыванию - нажатию F...(Khelp%)
'.............................................
   DIM SHARED NameFile$, NameRazd$, Npage1,
   KeyHelp
END
UsgtF1:
   IF NameFile$ <> "" THEN
      PCOPY 0, Npage1: KEY(KeyHelp) OFF
      ' отключить прерывание
      CALL HelpStoreScreen0(0)
      ' запомнить атрибуты курсора
      CALL HelpText(NameFile$, NameRazd$)
      PCOPY Npage1, 0: KEY(KeyHelp) ON
      ' включить прерывание
      CALL HelpStoreScreen0(1)
      ' восстановить атрибуты курсора
   END IF
RETURN
END

SUB HelpFNoff
   KEY(KeyHelp) OFF      ' отключить прерывание
END SUB

SUB HelpFNon (Nf$, Nr$, Nstr, Npage, Khelp)
'
'  инициализация режима Help
'  по нажатию клавиши F...
'  Nf$    -  имя текстового файла
'  Nr$    -  наименование раздела
'            ("" - вывод всего файла)
'  Nstr   -  номер строки, в которую помещается
'            сообщение <F1=Help>
'         = 0 - сообщение не выдается
'  Khelp  -  номер клавиши F...
'*********************************************
   KeyHelp = Khelp: Npage1 = Npage
   NameFile$ = Nf$: NameRazd$ = UCASE$(Nr$)
   KEY(Khelp) ON: ON KEY(Khelp) GOSUB UsgtF1
   IF Nstr > 0 THEN
      LOCATE Nstr, 1: COLOR 0, 7:
      PRINT "<F"+LTRIM$(STR$(Khelp))+"=Help>";
   END IF
END SUB
```

Листинг 2.
Два варианта реализации процедуры InkeyWaite — ожидание нажатия клавиши

INKEYW.BAS — с помощью оператора INKEY$
INKEYW45.BAS — с помощью системного прерывания

```
DEFINT I-N
'*********************************************

'**********  Kolesov QB Tools     *********
'*  БИБЛИОТЕКА подпрограмм QB-WAIT_KEY      *
'*********************************************
'*           Модуль INKEYW.BAS               *
'*        Ожидание нажатия клавиши           *
'*********************************************
END

SUB InkeyWaite (Kr$)
' ожидание нажатия клавиши
100   DO
200      Kr$ = INKEY$
300   LOOP WHILE Kr$ = ""
END SUB

DEFINT I-N

'*********************************************
'**********  Kolesov QB Tools     ***********
'*  БИБЛИОТЕКА подпрограмм QB-WAIT_KEY       *
'*********************************************
'*           Модуль INPINKEY.BAS             *
'*        Ожидание нажатия клавиши           *
'*     (ввод русской "р" для QB 4.5)         *
'*********************************************

' Замена функции INKEY$ с корректным вводом
' малой русской "р"
'
' Вызов:
'  CALL InkeyWaite(a$)
'
' Для работы этой процедуры необходимо
' подключение библиотеки QB.QLB, запуск
' среды надо выполнять:
'
'  QB.EXE /L
'.............................................
   TYPE Regtype
      ax AS INTEGER: bx AS INTEGER
      cx AS INTEGER: dx AS INTEGER
      bp AS INTEGER: si AS INTEGER
      di AS INTEGER: flags AS INTEGER
   END TYPE
   DIM SHARED inreg AS Regtype,
   outreg AS Regtype
'  DO: CALL InkeyWaite(a$): PRINT a$;: LOOP
END

SUB InkeyWaite (Kr$)
'*********************************************
' Ожидание нажатия клавиши и ввод символа
'
' Выход: Kr$ - код нажатой клавиши
' (1 или 2 байта)
'.............................................
' Конструкции:
'  CALL InkeyWaite(a$)
' и
'  DO: a$ = INKEY$: LOOP WHILE a$ <> ""
' идентичны, но 1-я корректно вводит
' малую русскую "р" на AT/286
' в системе QuickBASIC версии 4.5
'*********************************************
    ' ожидание ввода символа
    '.........................................
    ' компиляция с ключом /W —
    '  прерывание разрешено между строками
    '  100 и 200
Start:
100   DO
200   DEF SEG = 0: LOOP
      WHILE PEEK(&H41A) = PEEK(&H41C)
    ' дополнительная проверка на изменение
    ' адреса сегмента
   DEF SEG = 0: IF PEEK(&H41A) = PEEK(&H41C)
   THEN GOTO 100
   DEF SEG
    ' ввод символа
   inreg.ax = 0: CALL Interrupt(&H16, inreg,
   outreg)
    ' преобразование для однобайтового символа
   Kr$ = MKI$(outreg.ax)
   IF (outreg.ax AND &HFF) <> 0 THEN Kr$ =
LEFT$(Kr$, 1)
END SUB
```

Листинг 3.
Программа BC_VW.BAS — управление ключами /V/W при создании исполняемого модуля

```
DEFINT I-N
'*********************************************
'**********  Kolesov QB Tools     **********
'*          Программа BC_VW.EXE              *
```

```
'*******************************************
'*       Блокировка ключей /V, /W          *
'*     при запуске компилятора BC.EXE      *
'*******************************************
   ab$ = COMMAND$: a$ = ab$
   ' удаляем ключи /V,/W из командной строки
   in = INSTR(a$, "/V")
   IF in > 0 THEN
   a$ = LEFT$(a$, in - 1) + MID$(a$, in + 2)
   in = INSTR(a$, "/W")
   IF in > 0 THEN
   a$ = LEFT$(a$, in - 1) + MID$(a$, in + 2)
   COLOR 0, 7: PRINT " ----------
   Удалены ключи /V /W ----------";
   COLOR 7, 0: PRINT ""
   COLOR 0, 7: PRINT "Tab - повторить с
   исходными ключами";
   COLOR 7, 0: PRINT ""
   ' сброс клавиатуры
   DEF SEG = 0: POKE &H41A, PEEK(&H41C):
   DEF SEG
   ' запуск компилятора с отредактированной
   ' командной строкой:
   SHELL "g:\bc.exe " + a$
   s$ = INKEY$
   ' если была нажата клавиша TAB, то запуск
   ' компилятора с исходной командной строкой:
   IF s$ <> "" THEN
      IF ASC(s$) = 9 THEN
         COLOR 15, 4: PRINT " +++ повторить
         с исходными ключами +++";
         COLOR 7, 0: PRINT "":
         SHELL "g:\bc.exe " + ab$
      END IF
   END IF
END
```

Листинг 4.
Модуль MYMAKE.BAS — главный модуль программы оптимального управления режимом компиляции

```
DECLARE SUB TimeDateFileGet (FileName$,
Attribute%, ErrCode%, TimeDate&)
DEFINT I-N
'*******************************************
'**********  Kolesov QB Tools      **********
'*          Программа MYMAKE.EXE           *
'*******************************************
'*Запуск компилятора с проверкой времени   *
'*создания исходного и объектного файлов   *
'*******************************************
   CommandLine = COMMAND$
'
COLOR 0, 2
PRINT "--Запуск компилятора с контролем
времени коррекции .BAS и .OBJ--";
COLOR 7, 0: PRINT ""
'
' Выборка из командной строки
' имени исходного файла:
   FileName$ = CommandLine$:
   in = INSTR(FileName$, "/")
   IF in > 0 THEN
      FileName$ = LEFT$(FileName$, in - 1)
   ELSE
      in = INSTR(FileName$, ";")
      IF in > 0 THEN
   FileName$ = LEFT$(FileName$, in - 1)
   END IF
   FileName$ = LTRIM$(RTRIM$(FileName$))
   ' чтение даты создания исходного файла:
   CALL TimeDateFileGet(FileName$, 0, ErrCode%,
   TimeDate1&)
   IF ErrCode% > 0 THEN END    ' файл не найден
   ' формирование имени выходного файла
   OutName$ = FileName$
   DO
      in = INSTR(OutName$, "\")
      IF in > 0 THEN
      OutName$ = MID$(OutName$, in + 1)
      ELSE EXIT DO
   LOOP
   in = INSTR(OutName$, ".")
   OutName$ = LEFT$(OutName$, in) + "obj"
   ' чтение даты создания объектного файла:
   CALL TimeDateFileGet(OutName$, 0, ErrCode%,
   TimeDate2&)
   PRINT FileName$, OutName$, ErrCode%
   IF ErrCode% = 0 THEN        ' файл найден
   ' Надо производить перекомпиляцию?
   IF TimeDate1& < TimeDate2& THEN END ' нет
   END IF
   ' запуск компилятора
   SHELL "g:\bc.exe " + CommandLine$
END
```

Листинг 5.
Модуль CALLDIR.BAS — процедуры чтения имен и атрибутов элементов каталога через обращение к функциям DOS

```
DECLARE SUB FormDirNew (Filespec$,
FileMenu$(), Kfile%)
DECLARE SUB FormDirFirstSearch (Filespec$,
Attribute%, Ndta1%, Ndta2%, Nf%)
DECLARE SUB FormDirStart (Filespec$,
Attribute%, FileText$, Nf%)
OPTION BASE 1
DEFINT I-N
'*******************************************
'**********  Kolesov QB Tools      **********
'*  БИБЛИОТЕКА подпрограмм QB_STR_CONV     *
'*******************************************
'*          Модуль CALLDIR.BAS             *
'*  Процедуры, использующие функции DOS    *
'*      работы с файловой системой         *
'*******************************************
' ВНЕШНИЕ ССЫЛКИ: Загрузить библиотеку QB.QLB
'-------------------------------------------
' Состав процедур:
'
'   CALL FormDirStart -  чтение списка
            описателей элементов каталога
'   CALL FormDirNew   -  чтение списка
            имен файлов
'   CALL TimeDateFileGet -  чтение времени
            и даты создания/модификации файла
'.....................................
   CONST LenDiscr = 22  ' длина описателя файла
'
   TYPE RegTypeX
      ax AS INTEGER: bx AS INTEGER
      cx AS INTEGER: dx AS INTEGER
      bp AS INTEGER: si AS INTEGER
      di AS INTEGER: flags AS INTEGER
      ds AS INTEGER: es AS INTEGER
   END TYPE
   DIM SHARED inreg AS RegTypeX,
   outreg AS RegTypeX
END
SUB FormDirFirstSearch (Filespec$,
Attribute%, Ndta1, Ndta2, FileCount%)
'
' Первый поиск имени файла
'.....................................
' Вход:
'  Filespec$   - шаблон имени файла
'  Attribute%  - шаблон атрибута файла
'     маска:
'        &h01 - =1 - только для чтения
'        &h02 - =1 - спрятанный
'        &h04 - =1 - системный
'        &h08 - =1 - метка тома
'        &h10 - =1 - элемент подоглавления
'        &h20 - архив:
'               =1 - копия файла НЕ создавалась
' Выход:
'  FileCount% = 0 - найден первый
'     Ndta1, Ndta2 - адрес буфера DTA
'  FileCount% > 0 - не найден -
'     = КОД ОШИБКИ ПОИСКА
'  FileCount% = -1 - "Disk is not ready" -
'     только для BC 7.1
'     QB 4.5 - не обрабатывает такую ошибку,
'              происходит зависание программы
'.....................................
'  ON LOCAL ERROR GOTO DiskIsNotReady
'     для PDS 7.1
'------- Чтение адреса буфера DTA
   nam$ = Filespec$ + CHR$(0)
   inreg.ax = &H2F00:
   CALL interruptx(&H21, inreg, outreg)
   Ndta1 = outreg.es: Ndta2 = outreg.bx
'------- Первый поиск
   inreg.ax = &H4E00: inreg.cx = Attribute%
```

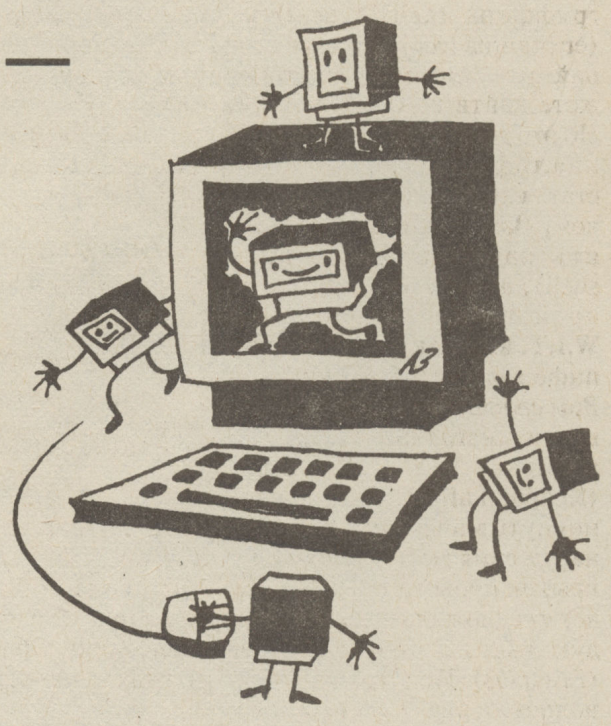
```
PDS7: inreg.ds = VARSEG(nam$)      ' для QB 4.5
    ' inreg.ds = SSEG(nam$)        ' для PDS 7.1
      inreg.dx = SADD(nam$)
      CALL interruptx(&H21, inreg, outreg)
      FileCount% = outreg.ax
' для PDS 7.1:
' EXIT SUB
' DiskIsNotReady: FileCount% = -1:
' RESUME QuitFD
' QuitFD:
END SUB

SUB FormDirNew (Filespec$, FileMenu$(),
Kfile)
'
' Чтение списка файлов с именами Filespec$
'-----------------------------------------
' Вход:
'  Filespec$ - шаблон имен файлов
'     Например: PATH\*.*, PATH\*.doc и прочее
'  Attribute% - шаблон атрибута файла
'     (см. процедуру FormDirFirstSearch)
'  Kfile - переменная
'     = -1 - перерезервируется -
'            REDIM FileMenu$(1 to Kfile)
'            и формируется массив FileMenu$()
'     >= 0 - запись имен файлов
'            в массив FileMenu$(kfile+1),
'            (kfile+2) и так далее.
'            (массив FileMenu$() - резервируется
'            в вызывающей программе)
' ВЫХОД (в любом случае):
'  Kfile - общая длина списка
'''''''''''''''''''''''''''''''''''''''''
   Attribute% = 0 ' чтение описателей только
                  ' простых файлов:
         '    "неспрятанных", "несистемных"
   CALL FormDirStart(Filespec$, Attribute%,
   FileText$, FileCount%)
   '
   ' IF FileCount% < 0 THEN -
   ' "Disk is not ready"
   IF FileCount% > 0 THEN
      IF Kfile < 0 THEN Kfile = 0:
      REDIM FileMenu$(1 TO FileCount%)
      ' перезапись в массив
      ks = 10       ' начало имени файла
      FOR i = 1 TO FileCount%
      ' выборка только имени файла в массив
         Kfile = Kfile + 1
         Lenn = INSTR(ks + 1, FileText$,
         CHR$(0)) - ks
         a$ = MID$(FileText$, ks, Lenn)
         FileMenu$(Kfile) = a$
         ks = ks + LenDiscr
      NEXT
   END IF
END SUB

SUB FormDirStart (Filespec$, Attribute%,
FileText$, FileCount%)
' Чтение списка файлов с именами Filespec$
'-----------------------------------------
' Вход:
'  Filespec$ - шаблон имени файла
'  Attribute% - шаблон атрибута файла
'            (см. процедуру FormDirFirstSearch)
' Выход:
'  FileCount% - количество найденных файлов
'     = -1 -   "Disk is not ready" -
'              для BASIC PDS 7.1
'     > 0:
'     FileText$ - список описателей файлов
'     каждый описатель - 22 символа:
'        1 - атрибут файла (1 байт)
'        2 - время создания/модификации
'            (2 байта)
'        4 - дата создания/модификации
'            (2 байта)
'        6 - длина файла (4 байта)
'        10 - имя файла (13 байтов в формате
'             "filename.ext"+CHR(0))
'''''''''''''''''''''''''''''''''''''''''
   CALL FormDirFirstSearch(Filespec$,
   Attribute%, Ndta1, Ndta2, FileCount%)
   IF FileCount% = 0 THEN
   ' выборка имен файлов
      FileText$ = "": f$ = SPACE$(LenDiscr):
      f1$ = ""
      Ndta21 = Ndta2 + &H15 - 1:
      DEF SEG = Ndta1
      DO
         FOR i = 1 TO LenDiscr
            MID$(f$, i) = CHR$(PEEK(Ndta21+i))
         NEXT i
         IF f$ = f1$ THEN EXIT DO
         f1$ = f$: FileCount% = FileCount% + 1:
FileText$ = FileText$ + f1$
         '------- Search next
         inreg.ax = &H4F00: inreg.ds = Ndta1:
         inreg.dx = Ndta2
         CALL interruptx(&H21, inreg, outreg)
      LOOP: DEF SEG
   ELSE IF FileCount% > 0 THEN FileCount% = 0
   END IF
END SUB

SUB TimeDateFileGet (FileName$, Attribute%,
ErrCode%, TimeDate&)
'
' Чтение времени и даты
' создания/модификации файла
'''''''''''''''''''''''''''''''''''''''''''
' ВХОД:
'  FileName$ - полное имя файла
' ВЫХОД:
'  ErrCode% - код ошибки
'  = 0 - найден
'     TimeDate& - время и дата
'     создания/модификации
'     TimeDate& = CVL(FileTime$ + FileDate$)
'        FileTime$, FileDate$ -
'           двухбайтовое двоичное представление
'           DOS значений времени и даты:
'        IntFileTime%=CVI(FileTime$)
'           значение разрядов:
'           0-4   - 2-секундные единицы (0-30)
'           4-10  - минуты (0-59)
'           11-15 - часы (0-23)
'        IntFileDate%=CVI(FileDate$)
'           0-4   - день (0-31)
'           5-8   - месяц (0-12)
'           9-15  - год (0-119,
'                   0 соответствует 1980 г.)
'
'  <> 0 - не найден
'''''''''''''''''''''''''''''''''''''''''''
   CALL FormDirFirstSearch(FileName$,
   Attribute%, Ndta1, Ndta2, ErrCode%)
   IF ErrCode% = 0 THEN
      '------- файл найден, определение времени
      ' и даты
      Ndta2 = Ndta2 + &H15: DEF SEG = Ndta1:
      f$ = SPACE$(4)
      FOR i = 1 TO 4:
      MID$(f$, i) = CHR$(PEEK(Ndta2 + i)):
      NEXT i
      DEF SEG : TimeDate& = CVL(f$)
   END IF
END SUB
```

Листинг 6.
Командный файл BC_RESET.BAT установки исходного режима компиляции

```
@echo off
if not exist mymake.exe goto mymakeset
if not exist bc_vw.exe goto bc_vwset

@echo Директория в исходном состоянии???
pause
goto end_reset
rem

:mymakeset
@echo Сброс режима MYMAKE.EXE
rename bc.exe mymake.exe
rename bc.old bc.exe
goto end_reset
rem

:bc_vwset
@echo Сброс режима BC_VW.EXE
rename bc.exe bc_vw.exe
rename bc.old bc.exe

:end_reset
```

«Hero Quest II» — игра для настоящих героев!

О. Стариков



Название игры: *Hero Quest II*
Фирма-разработчик: *SIERRA On-Line*
Звук: *Internal Speaker, Tandy, Sound Blaster, Adlib*
Управление: *клавиатура, джойстик, «мышь»*

«Hero Quest II», как и следует из ее названия, относится к традиционному для фирмы SIERRA On-Line классу Quest. Серия Hero Quest на сегодняшний день объединяет три серии с разными сюжетами, в которых действует один и тот же главный герой. Сегодня наш рассказ пойдет о второй части этой игры. Не обессудьте, господа читатели, но что имеем, о том и пишем. Эта интересная игра держит Вас в напряжении на протяжении всего времени, и очень нелегко оторваться от приключений в сказочной стране и вернуться к нашим серым будням. Привлекательность серии объясняется еще и тем, что для победы от Вас потребуется максимум терпения, смекалки и хитрости. И только тогда Вам удастся победить зло и попасть на прием к султану.

Компьютерные игры, так же как и все остальное программное обеспечение, не стоят на месте. И SIERRA вынуждена вместе со всем остальным миром двигаться вперед, в результате «Hero Quest» начинает приобретать, хотя и зачаточные, черты Role Playing Games. Вы имеете возможность в начале игры сформировать основные черты и наклонности Вашего героя. Наше изложение будет ориентировано на персонаж класса МАГ (WIZARD).

Сценарий игры

В самом начале Вы попадаете в город Шапэ (Shapeir), расположенный в сердце обширной пустыни. Путешествие начинается в гостинице Катты (Kattas Tail Inn). Настоятельно рекомендуется полюбоваться достопримечательностями города и его жителями. Поговорив с неким Абдуллой, Вы узнаете, что очень полезным предметом для похода по городу является карта (map). Но чтобы приобрести ее (а торгует картами господин Али Чика — Ali Chicka), Вам необходимо обменять Ваши деньги на местную валюту. Этим занимается, как Вы догадываетесь, местный пункт обмена валюты. Но по-английски он называется не Currency Exchange Office, а просто moneychanger. Совершив выгодный обмен, Вы возвращаетесь назад и приобретаете карту и компас (compass), который понадобится Вам для путешествия по пустыне.

Не ленитесь побродить по городу. Вы узнаете много интересного и полезного для дальнейшей игры. Город состоит из четырех площадей (plaza), которые соединяются между собой лабиринтами улиц. На площадях обитают торговцы (merchants), аптека (apothecary), магазин магических принадлежностей (magic shop), дворец султана (palace), оружейная лавка (weapons shop), фехтовальный зал (guild hall) и фонтан. На улочках Вы найдете Институт повышения квалификации колдунов-любителей (Wizard Institute of Technology — W.I.T.), звездочета (astrologer), волшебницу Азизу (enchantress) и упоминавшийся уже пункт обмена валюты. Когда Вы будете путешествовать по пустыне, Вам

рекомендуется отыскать дервиша (dervish), гнездо гриффина (griffin nest) и заколдованное дерево (enchanted tree). Чтобы не топать по пустыне пешком, наймите лошадку местной породы, которую Вы сможете найти в конюшне перед городскими воротами. Но это дело опасное, и Вам не помешало бы повысить квалификацию. Чтобы стать настоящим волшебником, Вам необходимо освоить заклинания (magic spell), а для этого придется пройти курс обучения в W.I.T. Кое-какую полезную информацию об институте Вам сообщат Азиза и продавец из магазина «Колдунреактив» Кипон Лаффин (Keapon Laffin). Чтобы проникнуть в институт, используйте заклинания для обнаружения магических сил (DETECT MAGIC) и открытия прохода (OPEN SPELL). После этого Вам будет предложено выбрать себе наставника. Я рекомендую взять в преподаватели волшебника Эразмуса (Erasmus). Подготовка Ваша будет заключаться в выполнении ряда учебных задач.

ИГРЫ

Только тот, кто проявит максимум терпения, смекалки и хитрости, победит зло, станет настоящим героем и попадет на прием к султану.

Первая задача — **колокола**. Вам потребуется использовать заклинания DETECT MAGIC, FETCH и TRIGGER SPELL.

Второе задание — **борьба со стихиями** (elementals). Сначала Вы должны справиться с ветром, для этого понадобятся заклинания FETCH и LEVITATE SPELL. Затем наступает черед земли. В этом испытании Вы должны применить заклинание DETECT MAGIC, а затем два раза подряд CALM SPELL. После этого от Вас потребуют справиться с водной стихией, которая предстанет перед Вами в облике айсберга. Но здесь все понятно — лед надо растопить (два раза воспользуйтесь заклинанием FLAME DART), а затем расколоть (заклинание OPEN SPELL). И наконец, испытание огнем. Здесь Вам придется потрудиться, последовательно применив заклинания OPEN SPELL, CALM SPELL, FETCH и как заключительный аккорд — FORCE BOLT SPELL.

Веселится и ликует весь народ. Теперь Вы настоящий волшебник, и в качестве памятного подарка награждаетесь новым заклинанием REVERSAL SPELL.

После того как Вы ознакомитесь с городом и жителями, начинают буйствовать разрушительные силы природы, с которыми Вы справлялись в ходе учебы. Перед битвой с ними Вам надо подготовиться и узнать о стихиях как можно больше. Первым на очереди будет дух огня (fire elemental). О нем Вы узнаете от аптекаря, у которого надо будет купить ловушку для огня и горшок с маслом, который Вам понадобится в дальнейшем для смазывания дверей. Кроме того, Вам пригодится лампа, которую можно приобрести у старьевщика на площади у фонтана. Дух огня возникает на площади у гостиницы Катты. Используйте ловушку, и огонь последует за Вами на улицу. Там положите лампу на землю (use lamp) и полейте огонь водой (use water). Теперь Ваша лампа стала волшебной (magic lamp).

Спустя некоторое время (четыре дня) появится дух воздуха (air elemental), о котором Вам подробно расскажет Кипон Лаффин. Попросите у него горшок грязи (earth). Еще Вам понадобятся мехи (bellows), которые красуются на вывеске оружейной лавки. Увы, мехи не продаются, но Вы можете получить их в качестве приза в борьбе армрестлинг (armwrestling) с продавцом оружия. Но вот Вы справились и выходите на охоту за духом воздуха. Смерч буйствует на дворцовой площади. Кидая грязь (throw dirt), Вы должны приклеить смерч к полу. Теперь используйте мехи, и Вы сумеете пленить смерч, став обладателями очередного волшебного предмета (bellows with air elemental).

Теперь наступает очередь духа земли (earth elemental). О нем Вы узнаете у человека-льва по имени Rakeesh. На Фехтовальной площади Вам надо будет также купить вещмешок (cloth bag). Дух земли скитается по улицам недалеко от Фехтовальной площади. Примените против него заклинание FLAME DART SPELL, а то, что останется, соберите в мешок (use cloth bag). Теперь Вы стали обладателем волшебного предмета — cloth bag with earth elemental.

Последней угрозой городу является дух воды (water elemental), о котором Вы узнаете у Азизы. Вам понадобится бурдюк (waterskin), который Вы купите на Фехтовальной площади. Дух воды возникнет на площади с фонтаном. Положите на землю бурдюк (use waterskin), а затем используйте мехи (use bellows). Вы получаете последний необходимый волшебный предмет — waterskin with water elemental.

Вы справились со всеми стихиями и стали национальным героем. Благодарное купечество дарит Вам достойное султана украшение — булавку (pin).

Теперь Вы можете путешествовать дальше. Вы отправляетесь с караваном в город Расэ (Raseir), но прежде должны основательно подготовиться. Необходимо получить растворитель, о котором Вы узнаете у аптекаря Гарика (Harik). Для этого потребуется перо гриффина (пока гриффин спит, используйте заклинание LEVITATE SPELL и возьмите перо — take feather). Затем добудьте некий таинственный фрукт сострадания (fruit of compassion). Его Вы найдете у заколдованного дерева, о котором расскажет Азиза. Сначала полейте дерево (watering tree), расскажите о себе (tell about hero), посыпьте землей (give earth). Теперь расскажите о духе земли (tell about earth elemental), обнимите дерево (give a hug) и назовите его имя (Julanar). Берите фрукт и несите его Гарику, который приготовит Вам растворитель. Теперь Вы можете присоединяться к каравану и отправляться дальше, на борьбу со злом.

Прибыв в город, зайдите в таверну «Синий попугай» (Blue Parrot), поговорите с главарем местной мафии и отправляйтесь обследовать город. К счастью, планировка Расэ и Шапэ практически одинакова. Вернувшись вечером в таверну, Вы знакомитесь с одним из уголовников, который расскажет Вам о капитане городской стражи Кавине (Khaveen) и о статуэтке сокола (statue of falcon).

Но на следующий день вор оказывается за решеткой, а Вас просят помочь женщине покинуть город. Вы отдаете ей свою одежду и визу, а за это Вам обещают помочь проникнуть во дворец, вывесив платок на стену, и дарят зеркальце (mirror). Когда настанет ночь, прокрадитесь на площадь с фонтаном, куда выходит окно капитана Кавина. Дождитесь ухода стражи и, использовав заклинание OPEN SPELL, проникните в спальню, где в шкафу хранится необходимая Вам статуэтка. Осторожно подойдите (stealth) к шкафу, смажьте его дверцы маслом (use oil), с помощью заклинания OPEN SPELL откройте шкаф и, взяв статуэтку, так же осторожно покиньте помещение. Возвращайтесь в таверну «Синий попугай». Как только Вы ее покинете, Вас сразу же арестует местный ОМОН и отведет в тюрьму, где уже томится некий Катта. Показав ему булавку (pin), Вы завоевываете его доверие и с помощью заклинания OPEN SPELL открываете клетку, берете свои вещи и через потайной ход вместе с Каттой покидаете темницу.

Оказавшись в запретной части города, Вы встречаете колдуна Ада Ависа (Ad Avis), который с помощью телепортации переносит Вас в забытый город и приказывает, проникнув в склеп, добыть статую бога Иблиса (statue of Iblis).

С помощью зеркальца (use mirror) Вы входите в склеп, зажигаете лампу (use lamp) и начинаете преодолевать одно препятствие за другим. Вначале Вам будет преграждать путь быстрый водный поток, по которому проплывают бревна. Используйте их для переправы на противоположный берег. Но на другой стороне реки из отверстия дует сильный ветер. Не беда, Вам поможет заклинание FORCE BOLT SPELL. Следующим препятствием будет раскаленная лава. Облейтесь водой (use water) и смело продолжайте путь. Наконец Вы окажетесь на вершине утеса, у подножия которого находится дверь. Спустившись к ней и произнеся заклинание LEVITATE SPELL, Вы услышите вопрос стражи сокровища. Пароль для входа — «Suleiman». Проникнув в комнату, идите, ничего не трогая. И вот Вы достигли статуи. Но неожиданно появляется гнусный Ад Авис, забирает ее, и, засыпав камнями выход, оставляет Вас на верную погибель в пещере. Однако, внимательно осмотрев комнату, Вы обнаруживаете на сталагмите кольцо. Надев его, выпускаете джинна, который по старой доброй традиции предлагает Вам за свое освобождение выполнить три желания. Первые два можете использовать для улучшения своих бойцовских качеств, а третье — для телепортации в город Расэ.

Джинн переносит Вас на дворцовую площадь, и Вы встречаете Катту, который в это время отвлек дворцовую стражу. Прокравшись ко дворцу, поднимитесь на стену (cast levitate) со стороны висящего платка и, минуя евнухов, пробирайтесь во дворец. Очутившись на балконе, избавьтесь от капитана Кавина с помощью заклинания CALM SPELL, а затем, использовав заклинания LEVITATE SPELL и OPEN SPELL, спускайтесь и открывайте дверь, за которой Вы встретите старого знакомого Ада Ависа. Действовать надо быстро и решительно. Вначале необходимо справиться со статуей. Начинайте с заклинаний TRIGGER SPELL и REVERSAL SPELL, затем метким броском кинжала (throw dagger) сбейте еще не зажженную свечу. Теперь отбегите в сторону и киньте кинжал в кадило, висящее на треноге, но так, чтобы оно опрокинулось на Ада Ависа. Если Вы успели все это сделать, то победа в Ваших руках.

Теперь осталось только принять все почести, которых Ваш герой безусловно достоин. Но лучше Вам взглянуть на это своими глазами!

Управление игрой

Вы можете использовать как клавиатуру, так и «мышь». Перемещение героя осуществляется с помощью клавиш управления курсором. Команды выполняются путем их ввода в командную строку экрана. С помощью «мыши» вызывается верхнее меню, а также наводится цель при бросании предметов и использовании магических заклинаний. Комбинацией клавиш можно также вводить команды.

Теперь Вы можете отправляться в интересный и занимательный мир игры, где добро побеждает зло, а простые люди становятся известными героями. Основные проблемы я осветил, но существует еще достаточно много трудностей на Вашем пути к славе. Например, улучшение основных качеств героя, постоянные сложности с финансированием героической деятельности и так далее. Помните, что за каждое нужное и правильное действие или вопрос Вам начисляются очки. А сейчас в путь! Слава, признание и приключения ждут Вас! Желаю удачи. ■

Системный лабиринт!

В этой игре Вы найдете:

- *антураж средневекового замка;*
- *великолепие различных видов старинного оружия;*
- *EGA-графику и анимацию высокого разрешения.*

Игра относится к серии игр интеллектуального характера по тематике ТРИЗ (теория решения изобретательских задач). Предназначена для развития системного мышления и применения в общеобразовательных школах, вузах и школах бизнеса.

Для получения игры перешлите 15$ (в рублях по текущему курсу) на счет НТО «Софт-Москва»: р/с 467996 в Независимом Банке России, к/с 1161385 в РКЦ-2 ГУ ЦБ РФ по г. Москве, МФО 201779.

Не забудьте указать тип дискеты: 5.25" или 3.5"!

Анонс

«Монитор» —версия 1.94

Д. Мастрюков. Алгоритмы сжатия информации. Часть 2. Арифметическое кодирование
Арифметическое кодирование, как правило, дает более высокую степень сжатия, чем кодирование по Хаффмену, однако требует большего объема вычислений.

А. Карев. Программирование контроллера накопителя на гибких магнитных дисках
Подробное описание устройства и правил программирования контроллера НГМД.

В. Плотников. Использование вирусной технологии для создания демонстрационных программ
Рассматриваются вопросы внесения изменений в исполняемые модули задач для получения демонстрационных примеров.

О. Шарапов. Загружен — не загружен, загружен — не загружен, загружен...?
Рассматривается проблема определения повторной загрузки резидентной программы в память компьютера.

А. Тихонов. Пролог еще жив...
Обзор современных систем программирования на языке Пролог.

Журнал «Монитор» можно приобрести в городах России, Украины, Беларуси, Узбекистана, обратившись к нашим официальным дилерам.

1. Самара, МП «Сети»,
директор Дмитрий Грецов
Тел. (8462) 354-932
2. Волгоград,
арендное предприятие «Техническая книга»
Тел. (8442) 363-597
3. Чебоксары, фирма «Демос»
Тел. (8352) 208-395, 426-492, факс 209-437
4. Киев, МП «Диас»
Тел./факс (044) 220-77-15,
тел. 518-80-24
5. Днепропетровск, Попов А.Е.
Тел. (0562) 428-593
6. Минск, НПП «Триумф»
Тел. (0172) 666-335,
факс 666-606
7. Ташкент, Информ-Проспект
Тел. (3712) 543-955
факс 458-593
8. Тамбов, АО «От Елены»
ЛПС «Shift»
Тел. (0752) 22-44-63
9. Пермь, Попов А. Л.
Тел. (3422) 33-30-91
10. Иркутск, Мадьяров И. И.
Тел. (3952) 46-19-68

ELITSoft

Пользователям вычислительной техники
Акционерное общество «ЭлитСофт»
предлагает

Агрегаты Бесперебойного Питания (Uninterruptable Power Source)

Основные характеристики

- *Мощность* 900 ВА
- *Время автономной работы* 40 мин.
- *Время переключения* 5-7 мс
- *Рабочий диапазон* 180-240 В
- *Гарантия* 1 год
- *Послегарантийное обслуживание*

В конструкции предусмотрены

- *Фильтрация сетевых помех*
- *Звуковая и световая сигнализация*
- *Индикация перегрузки*
- *Индикация уровня заряда батарей*
- *Герметичные необслуживаемые аккумуляторные батареи*

Поставка и оплата

Поставка со складов в Москве, Санкт-Петербурге, Чите, Рязани.

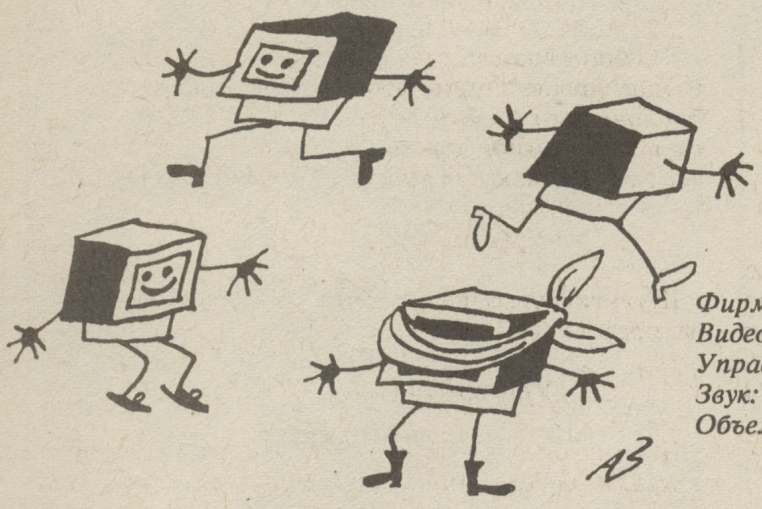
Любая удобная Вам форма оплаты.

Адрес: 117924 Москва, ул. Губкина, д. 3, корп. «Г»
Телефон: (095) 132-6904, 132-6905
Факс: (095) 132-6792, 135-3040

Мы продаем компьютеры, модемы и факс–модемы производства США!

Там на неведомых дорожках...

К. Бакай



Фирма-разработчик: C.V.S.(1991г.)
Видеоадаптер: VGA, EGA, MCGA
Управление: «мышь», клавиатура
Звук: PC Speaker, Adlib, Sound Blaster
Объем на диске: 1200 Кб

Предполагаю, что многие из вас сталкивались с этой, на первый взгляд, непонятной и в то же время притягивающей своим юмористическим оформлением и какой-то загадкой игрой — «Гоблины».

Но, промаявшись с паролем и наконец-таки после 101-й попытки (если у Вас хватило терпения и сил) попав в первый экран, Вы теряетесь, мучимые догадками, как управлять и что делать?

Для этого необходимо вникнуть в суть дела. Как Вы уже поняли из вступительного мультика, какой-то нехороший дядя-колдун решил испортить праздник жизни местному королю гоблинов путем изготовления куклы (известной в среде любителей компьютерных игр как wodoo doll) и с помощью черной магии погубить его. Но есть, есть еще в жизни место подвигу. Сын короля принц Буффал, придворный маг и королевский шут решили воспрепятствовать силам зла, и поэтому отправились в свой «крестовый» поход супротив колдуна. Каждый из этой отважной команды выполняет свою функцию. Принц не обладает особым излишком ума, но восполняет этот пробел кулаками, которые по Вашей команде он готов применить всегда и везде. Маг на вид добродушный старичок, но стоит ему поводить руками над какой-нибудь ерундой, как она превращается в нужную вещь. И последний из компании, но самый умный и крайне необходимый шут может подбирать и использовать по назначению различные предметы, попадающиеся в пути.

Управление осуществляется как «мышью», так и стрелками клавиатуры. Выбор персонажа — клавишей «пробел», выбор функции — клавишей Esc. Курсор в виде кулака указывает выбранному персонажу место для применения своих способностей, то есть маг будет колдовать, принц работать кулаками, а шут при наличии у него какой-нибудь вещи использовать ее по своему усмотрению. Курсор в виде стрелки поможет Вам перемещать наших маленьких героев по экрану. Кроме этого, шут умеет поднимать предметы (при этом курсор принимает форму кисти руки), но запомните: карман его вмещает только одну вещь. Сигналом к исполнению служит клавиша Enter.

Как видите, особых сложностей с освоением управления игрой не предвидится, а посему, немного потренировавшись, Вы можете смело выдвигаться на передовые рубежи борьбы за всеобщую победу гоблинизма!

В самом начале игры Вы находитесь у входа в скромную обитель колдуна, заколдовавшего короля. Вы должны попытаться проникнуть в помещение и договориться, что называется, по-хорошему. Как видите, идея мирного решения конфликта противостоящих сторон, в общем-то, не нова.

Уровень 1

Здесь Вы сразу же убедитесь, что в дверь так просто не пускают. Поэтому действуем по принципу известного и любимого детишками героя — Бармалея, то есть идем в обход. Для этого попросим принца стукнуть по каменной арке, в результате чего у изваяния отвалится рог. Его подберет шут (с Вашей, конечно же, помощью). Затем подведите шута к створу ворот и подуйте в свою находку. Упавшую с дерева ветку маг превратит в кирку, которая крайне необходима шуту. После этой процедуры в левом нижнем углу появится код следующего уровня, который Вы можете использовать при дальнейших входах в игру.

Уровень 2 (код=UQUQFDE)

Пропасть нужно засыпать яблоками, которые увеличиваются в объеме, если маг поколдует над ними, а затем принц собьет кулаками. Шут, вы-

полнив операцию по засыпке пропасти, выковырнет киркой алмаз из скалы и спрячет находку в карман. Годятся только первое и третье справа яблоки.

Уровень 3 (код=ICIGCAA)

Все очень просто — достаточно шуту постучать в дверь.

Уровень 4 (код=ECPQPCC)

Необходимо отдать колдуну алмаз. Для этого маг колдует над ближайшим цветком, принц по стеблю забирается на стол. Шут, положив на время алмаз на пол, забирается на другой стол, берет первый горшок (pot 1) и отдает его содержимое второму цветку. Путь свободен, но наш бедный малыш не умеет лазать. На помощь приходит головорез принц, столкнув книгу, по которой шут влезает на стол и отдает колдуну алмаз.

Ну вот, вроде бы мы и пришли к консенсусу. Колдун соглашается освободить от заклятия короля, но Вы должны принести ему три волшебных предмета. Что же, попробуем выполнить условия договора.

Уровень 5 (код=FTWKFEN)

Первым идет маг и колдует над ростком возле страшилы. Принц, взобравшись по выросшему стволу, бьет в глаз каменному истукану, на высунувшийся язык надо поставить шута, оставив место для мага. Теперь маг направляет свои чары на основание гроба, после пятой попытки надо бежать на язык истукана, иначе будет худо (душераздирающее зрелище!). Когда все «пассажиры» в сборе, принц бьет еще раз в глаз, и друзья спасены. Третий удар опустит их на землю.

Уровень 6 (код=HQWFTFW)

Маг колдует над пауком. Принц взбирается по паутине наверх и, дернув за свисающий конец, приподнимает насекомое. Шут берет револьвер и убивает поочередно правого и левого пауков. Затем, взяв подушку, подкладывает ее под центрального паука, над которым колдует маг. Теперь осталось только поднять упавшую бутыль.

Уровень 7 (код=DWNDGBW)

Только после того как маг поколдует над мешочком с семенами, шут может взять его и засеять поле. Принц, избивая пугало, спасет урожай от птиц. Полив организует маг, воздействуя на вторую тучку. Но стоит только сорвать выращенный цветок, как появляется колдун и требует всё собранное с таким трудом. Ничего не поделаешь, уговор дороже денег.

К сожалению, колдун полагает иначе. В результате Ваши друзья оказываются в заточении в жуткой и ужасной пещере. И, естественно, король по-прежнему заколдован. Но не все еще потеряно!

Уровень 8 (код=JCJCJHM)

Если шут возьмет вазу и поставит ее возле скелета, то магу можно начать работу. Поколдовав над скелетом, получим флейту, играя на которой, шут открывает путь наверх принцу. Тот бьет по камню над помостом, используя доску в качестве трамплина.

Уровень 9 (код=ICUGCGT)

Шут, тихонько прокравшись мимо эдакого Цербера, крадет кусок мяса и показывает его собаке.

Уровень 10 (код=LQPCUJV)

Шут, показав кусок мяса кому-то, спрятавшемуся в дупле, берет сачок. Маг своим искусством выращивает сук из сучка, и, используя эту примитивную катапульту, принц закидывает своих коллег на следующий ярус. Там маг, поколдовав над гнездом, превращает его в пробку, которой шут затыкает дупло. Затем принц пугает птичку, а шут ловит ее у второго дупла сачком.

Уровень 11 (код=HNWVGKB)

Шут, подкравшись к собаке, выпускает птичку. Да, какая собака не мечтает стать птицей. Маг помогает ей: пес превращается в птицу. Принц, постучав в боковую дверь, открывает проход на следующий уровень.

Уровень 12 (код=FTQKVLE)

Взяв перо из чернильницы, шут может пощекотать скелет, и если подать скелету чашку с мячом, то можно смело брать ключ и открывать клетку. Узник, которого Вы освободите, в дальнейшем поможет Вам. Маг превратит перо в мухобойку, которой можно убить надоедливую осу. Поколдовав над ее трупом, получим дротик, который просто необходимо метнуть в портрет ненавистного колдуна. Реакция просто потрясающая: выпадает кукла короля. Не забудьте прихватить с собой эликсир, который делает шута невидимым.

Уровень 13 (код=DCPLQMH)

Маг колдует над корягой. Шут подбирает дудку и, взобравшись на ветку, дует в нее. На зов прилетает птица и высиживает яйцо. Разбить скорлупу должен принц, а взрастить птенца — маг. Его переносят в безопасное место, где, пустив в ход свои чары, он расчищает путь шуту. Открытый участок местности он может преодолеть, выпив предварительно эликсир (все зависит от Вашей сноровки и координации

движений). И вот шут и маг уже вдвоем. Шут возьмет морковку и покажет крысе, которой займется колдун. Пока он отвлекся, проведите принца к друзьям.

Уровень 14 (код=EWDFNNI)

Маг превращает маленький камешек в лесенку, по которой шут, предварительно взяв палку, взбирается на плиту. Вставив палку в отверстие, он получит лейку. Теперь надо полить кустики. Собирать урожай позовите мага. Первая морковка восстановит Вам жизнь, третья откроет дверь, пятая даст ключ, который выбьет принц. Откройте дверь.

Уровень 15 (код=TCNGTOV)

Принц подготавливает пушку к стрельбе и выбивает ядро из кучки. Шут заряжает пушку. Принц прицеливается. Возьмите спички и подожгите фитиль. Аналогичную операцию произведите с упавшей морковкой: принц должен прицелиться вновь. Попытайтесь попасть в котел и разведите огонь под ним. Запах супа разбудит хозяина, но, увы, он плохо слышит. Поэтому добудьте еще одну морковь. Поколдовав над ней, Вы получите рупор, через который можно поговорить со старичком. Палочкой, которую он даст, надо ударить в гонг. Появившийся маятник возьмите с собой.

Уровень 16 (код=TCVQRPM)

Перенесите камень на указанное место и поколдуйте над ним. Поднявшись на плавучий остров, примените чары. Полученную в результате превращения кирку принц скинет вниз. Шут с помощью маятника определит место входа в подземелье и киркой откроет проход.

Уровень 17 (код=IQDNKQO)

Маг поколдует над мешочком, и шут может его забрать. Принц выбивает из поленницы бревнышко, шут применяет его против капкана, а маг превращает его в дезодорант. Шут, используя все это по порядку, ловит бесхозную ногу и, положив перед драконом, поджаривает ее. Данный «шашлык» бросьте на шахматную доску. Возьмите меч и подставьте его под пламя дракона. Хватайте получившийся в результате всех этих манипуляций факел и смело выходите на поверхность земли.

Уровень 18 (код=KKKPURE)

Шут подходит к статуе и зажигает «вечный огонь» в груди, затем поднимается на правую ладонь и светит факелом. Подберите ключ и вставьте его в левое ухо титана. Теперь по одному взбирайтесь на правую ладонь и телепортируйтесь в город колдуна.

Уровень 19 (код=NGOGKSP)

Принц сбивает банан. Задача шута — задобрить писаря и получить у него книгу заклинаний (Witchcraft), используя поочередно банан, фальшивый нос и мыло. Настоятельно рекомендую другие книги не читать. Как говорится, излишняя грамотность вредит! Маг откроет верхнюю решетку.

Уровень 20 (код=NNGWTTO)

Маг превращает пенек в рычаг, дернув за который, принц опускает лесенку, и, поколдовав над ухом джинна, прочищает ему слуховой аппарат. Шут тут же зачитывает свое заклинание, берет карпа и несет его на чердак, оттуда забирает миску. Необходимо эту миску поставить около джинна и снова прочитать заклинание ему на ухо, после этого взять миску и вылить ее содержимое на чудо-юдо. После этого возьмите рогатку и снова в путь.

Уровень 21 (код=LGWFGUS)

Шут стреляет по вязанке бананов, освобождая для принца проход к рычагу. Из открывшейся решетки выплывает рыба. Усаживайтесь на нее поудобнее и плывите по этому потоку дальше.

Уровень 22 (код=TQNGFVC)

Маг превращает плитку в лесенку, взобравшись по которой, шут перебьет у основания веревку и закинет ее на верхний коготь птицы. После этого объявляем колдуну последний и решительный бой. По причине отсутствия танковой и артиллерийской поддержки стреляем из рогатки. Колдун превращается в приятную пичужку, которая хватает шута и скрывается с ним под потолком. Приходится на помощь звать мага. Но в результате воздействия чар птичка-невеличка превращается в черепаху-попрыгунчика, которая выпускает шута и хватает мага. Теперь наступает очередь принца. Мощнейший удар правой — и маг освобожден. Но враг хитер и коварен! Колдун превращается в паука и запутывает в паутине принца. Снова воспользуемся рогаткой, на этот раз чтобы освободить принца, а затем маг колдует над пауком. Паук распадается на три источника и три составные части, они должны попасть в мешок, который шут заблаговременно перенесет поближе к рыбе. Как только операция завершена, шут хватает мешок с колдуном и несет его к котлу.

И теперь в королевстве гоблинов народные гуляния, мерзкий колдун пойман и посажен в бутылку. Но коварный враг не дремлет! Уже вышла вторая часть игры, и мы постараемся в ближайшее время описать и ее тоже.

Все на борьбу за светлое будущее карапузов-гоблинов! ■

Откуда ты, моя программа?

Б. Горбунов

Если Вы хотите хранить параметры Вашей программы в отдельном файле, то Вам, возможно, потребуется определить маршрут доступа к ней, когда файл параметров предполагается хранить в том же каталоге, из которого загружена программа. Алгоритм определения маршрута представлен в программе на языке Turbo Pascal (пример). Этот алгоритм действует в среде DOS, начиная с версии 3.0. Исходную информацию для построения алгоритма можно почерпнуть в [1] и [2].

При запуске каждая из программ получает свой префикс программного сегмента (PSP), в котором среди параметров хранится также и сегментный адрес копии командной среды DOS для запускаемой программы (слово со смещением 2Ch от начала PSP). В командную среду DOS входят строки, завершаемые нулевыми байтами, заканчивается последовательность строк дополнительным нулевым байтом, вслед за которым идет слово, хранящее количество дополнительных строк командной среды (обычно имеет значение 1), а затем строка, содержащая полное имя файла, из которого была загружена программа.

В модуле System языка Turbo Pascal объявлена специальная переменная PrefixSeg:word, содержащая сегментный адрес PSP программы. Сегментный адрес PSP можно получить также, воспользовавшись функцией 62h прерывания 21h DOS, которая возвращает его в регистре BX. ■

Литература

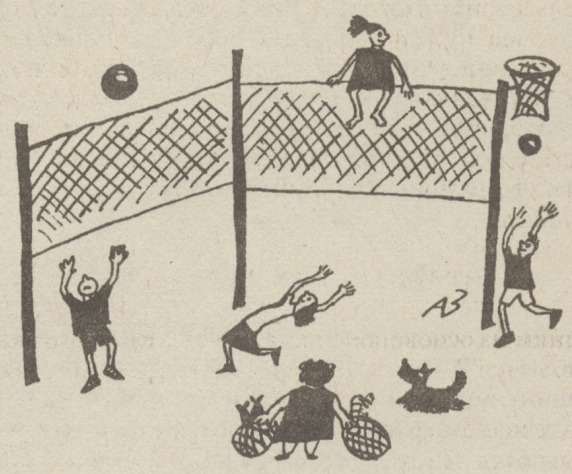
1. **Оператору ЭВМ. DOS and BIOS.– М.: Наука, 1990.**
2. **Бородич Ю.С. и др. Паскаль для персональных компьютеров: Справ. пособие.– Минск: Вышейшая школа, БФ ГИТМП «Ника», 1991.**

Пример.
Программа на языке Turbo Pascal, демонстрирующая алгоритм определения маршрута доступа к файлу, из которого она загружена

```
program StartDir;

uses
  Dos;

type
  DosEnvArray = array [0..MaxInt] of char;

var
  ProgramDir    : DirStr;
  DosEnv        : ^DosEnvArray;
  DosEnvPos     : integer;
  ProgramExt    : ExtStr;
  ProgramName   : NameStr;
  ProgramPath   : PathStr;
  PathStrPos    : integer;
  PreviChar     : char;

Begin
  DosEnv := Ptr (MemW[PrefixSeg:$002C],$0000);
  DosEnvPos := 0;
  PreviChar := #255;
  while ((DosEnv^[DosEnvPos] <> #0) or (PreviChar <> #0))
      Do begin
          PreviChar := DosEnv^[DosEnvPos];
          Inc (DosEnvPos);
      end;

  Inc (DosEnvPos,3);
  PathStrPos    := 1;
  repeat
      ProgramPath[PathStrPos] := DosEnv^[DosEnvPos];
      Inc (PathStrPos);
      Inc (DosEnvPos);
  until (DosEnv^[DosEnvPos] = #0);
  ProgramPath[0] := char (PathStrPos - 1);

  FSplit  (ProgramPath,ProgramDir,
           ProgramName,ProgramExt);
  WriteLn ('      Полное имя файла, из которого загружена
                  программа:');
  WriteLn ('''' + ProgramPath + '''');
  WriteLn ('      Маршрут доступа к файлу:');
  WriteLn ('''' + ProgramDir + '''');
  WriteLn ('      Имя файла:');
  WriteLn ('''' + ProgramName + '''');
  WriteLn ('      Расширение имени файла:');
  WriteLn ('''' + ProgramExt + '''');
End.
```

■

Интервальные методы в компьютерной графике



С. Курковский

Введение

В данной работе рассматриваются оригинальные методы построения линий уровня и поверхностей различных функций. Среди множества различных методов рассматриваемые далее алгоритмы отличает то, что на вид графика не накладывается никаких ограничений, более того, исследуемая функция может быть задана в неявном виде, то есть в виде f(x,y,z)=0. Такой результат достигается за счет применения *интервальных методов*. Среди прочих работ, посвященных интервальным методам (как, например, [1]), данную работу отличает также яркая практическая направленность. В разделе «Графические алгоритмы» кратко рассказывается об общих чертах методов построения графиков, линий уровня и поверхностей. Раздел «Интервальная арифметика», как следует из его названия, даст основные понятия и определения из теории интервалов, необходимые для дальнейшего обсуждения. В разделах «Алгоритмы построения линий уровня» и «Построение трехмерных поверхностей» приводятся три практических алгоритма, использующих интервальную арифметику. Следующий раздел, как и следует из его названия — «Практическая реализация интервальных методов», полностью посвящен программированию описанных методов на языке C++.

Графические алгоритмы

Практически все алгоритмы построения графических отображений математических функций, как правило, состоят из двух этапов. Первый этап заключается в отыскании линии или поверхности (далее — *контура*), второй этап состоит в изображении найденного контура. Очевидно, что первый этап является наиболее важным.

В целях отыскания нужного контура часто используется так называемый *метод пробных точек*, заключающийся в вычислении знака функции в углах отображаемой прямоугольной области. Если эти знаки неодинаковы, то это свидетельствует о том, что контур проходит через данную область. Основная проблема заключается в том, что этот способ не может гарантировать обнаружения контура. На рис. 1 приведены ил-

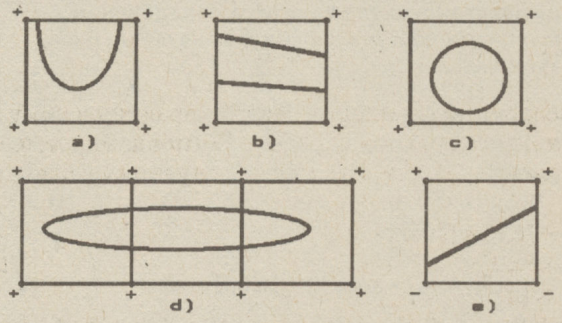


Рис. 1. Возможные ситуации при применении метода пробных точек
Черные точки показывают места вычисления знаков функции.
a) Контур дважды пересекает границу квадранта.
b) Квадрант пересекают одновременно два участка контура.
c) Маленький замкнутый контур целиком лежит внутри квадранта.
d) Комбинация ситуаций a и b.
e) Обнаруживаемый участок контура дважды пересекает границу квадранта.

люстрации различных ситуаций, возникающих при использовании метода. Таким образом, метод пробных точек нуждается в некоторой дополнительной информации, чтобы гарантировать обнаружение контура. Не вдаваясь в математические подробности, можно сказать, что одной из самых эффективных и вместе с тем простой возможностью для получения этой информации является применение *интервального анализа.*

Интервальная арифметика

Одним из основоположников интервального анализа является R.E. Mooge. В его работе [2] развиваются основные положения теории интервалов.

Интервалом $I=[a,b]$, $a \le b$ называется множество вещественных чисел, заключенных в отрезке $[a,b]=\{x \mid a \le x \le b\}$.

Если A и B являются интервалами и @ — одна из арифметических *операций* +, −, × или /, тогда A@B определяется как

$$A@B = \{x@y \mid x \in A,\ y \in B\} \qquad (1)$$

Вещественное число a можно рассматривать как интервал $a=[a,a]$. Таким образом, определение (1) описывает и выражения вида:

$A*a$
$a+A$
A/a
$(-1)\times A = -A$

Определение (1) неприменимо к практическому исчислению интервалов. В работе [2] показывается, что определение (1) эквивалентно следующим правилам.

$$\begin{aligned}
&[a,b]+[c,d]=[a+c,\ b+d];\\
&[a,b]-[c,d]=[a-d,\ b-c];\\
&[a,b]\times[c,d]=\\
&\quad=[\min(a\cdot c,\ a\cdot d,\ b\cdot c,\ b\cdot d),\ \max(a\cdot c,\ a\cdot d,\ b\cdot c,\ b\cdot d)];\\
&[a,b]/[c,d]=[a,b]\times[1/d,\ 1/c]
\end{aligned} \qquad (2)$$

при условии, что $0 \notin [c,d]$.

Кроме того, к интервалам применимы и такие операции теории множеств, как *пересечение* и *объединение*:

$$\begin{aligned}
&A \cap B = \{x \mid x \in A \text{ и } x \in B\};\\
&A \cup B = \{x \mid x \in A \text{ или } x \in B\}.
\end{aligned} \qquad (3)$$

Данное выше правило деления интервалов может быть расширено с помощью определения (3) в случае, когда $0 \in [c,d]$. Как доказывается в [2], в этом случае результирующий интервал будет интервалом или объединением интервалов с бесконечными границами:

$$\begin{aligned}
&[a,b]/[c,d]=\\
&\quad=[b/c,+\infty) \text{ если } b\le 0 \text{ и } d=0;\\
&\quad=(-\infty,\ b/d]\cup[b/c,+\infty) \text{ если } b\le 0,\ c<0 \text{ и } d>0;\\
&\quad=(-\infty,\ b/d] \text{ если } b\le 0 \text{ и } c=0;\\
&\quad=(-\infty,\ a/c] \text{ если } a\ge 0 \text{ и } d=0;\\
&\quad=(-\infty,\ a/c]\cup[a/d,+\infty) \text{ если } a>0,\ c<0 \text{ и } d>0;\\
&\quad=(-\infty,+\infty) \text{ если } a<0 \text{ и } b>0.
\end{aligned} \qquad (4)$$

Кроме того, как следует из определений (4), интервал $[a,b]/0=(-\infty,+\infty)$.

ГРАФИКА

Среди интервальных методов построения трехмерных поверхностей можно выделить два: метод построения линий уровня функции на параллельных осям координат плоскостях и метод разбиения трехмерного пространства.

Рассмотрим теперь, как можно применить теорию интервалов к исследованию рациональных функций. *Рациональной функцией* называется функция, которая может быть описана с использованием конечного числа операций +, −, × и /. Отношение полиномов — также рациональная функция. Для данной рациональной функции $f(x_1,x_2,\ldots,x_n)$ можно построить ее *интервальное расширение* $F(X_1,X_2,\ldots,X_n)$, где каждая переменная x_i определена на интервале X_i, то есть $x_i \in X_i$. Интервальное выражение, описывающее функцию F, вычисляется с использованием определений (2) и (4), в результате чего также получается интервал или объединение интервалов. Полученный интервал является областью изменения функции f при $x_i \in X_i$. Это непосредственно следует из основного свойства интервальной арифметики: $x \in X \Rightarrow f(x) \in F(X)$. Приведем простой пример: пусть $f(x,y)$ — функция двух переменных, x и y определяют прямоугольную область $a \le x \le b$, $c \le y \le d$ на плоскости (x,y). Вычислив интервальное расширение $F(X,Y)$ функции $f(x,y)$, где $X=[a,b]$ и $Y=[c,d]$, можно выяснить, что область изменения функции $f(x,y)$ лежит на интервале $F(X,Y)$. Границы области изменения функции, определяемые концами $F(X,Y)$, могут быть неточными, но при этом *гарантируется*, что все значения функции $f(x,y)$ лежат на этом интервале.

Для функций, не являющихся рациональными, также может быть построено интервальное расширение. Для монотонных функций, таких как e^x, $\ln x$, $\sqrt{x}$ на интервале $X=[a,b]$ их интервальные расширения выглядят следующим образом:

e^x : $EXP(X)=[e^a,\ e^b]$;
$\ln x$: $LN(X)=[\ln a,\ \ln b]$;
$\sqrt{x}$: $SQRT(X)=[\sqrt{a},\ \sqrt{b}]$

Не намного сложнее вычисление интервальных расширений для немонотонных функций, например, тригонометрических, поскольку их экстремальные точки (максимумы и минимумы) хорошо известны.

Алгоритмы построения линий уровня

На рис. 2 показан очень простой алгоритм для рисования линии уровня функции двух переменных. Идея заключается в рекурсивном разбиении исходной прямоугольной области (квадранта) на четыре квадранта меньшего размера до тех пор, пока не будет достигнут некоторый искомый размер квадранта, в котором будет нарисована линия. Этот алгоритм не использует интервальную арифметику, но на его основе будут построены другие, более сложные *интервальные* алгоритмы.

```
Процедура  Линия_Уровня (Глубина : Глубина_деления;
           X, Y : Точка;
           dX, dY : Расстояние;
           F : Функция)

   Процедура  Разделить_Квадрант
   {
      Линия_Уровня (Глубина+1, X,      Y,      dX/2, dY/2, F);
      Линия_Уровня (Глубина+1, X+dX/2, Y,      dX/2, dY/2, F);
      Линия_Уровня (Глубина+1, X+dX/2, Y+dY/2, dX/2, dY/2, F);
      Линия_Уровня (Глубина+1, X,      Y+dY/2, dX/2, dY/2, F);

{
   Если  Глубина < Искомая_Глубина  то  Разделить_Квадрант;
   Иначе  Рисовать_Линию_Уровня (X, Y, dX, dY, F);
}
```

Рис. 2. Алгоритм 1 для рисования линии уровня

Процедура **Линия_Уровня** вызывается с параметром **Глубина=0**, что соответствует исходному неразделенному квадранту, точка (X,Y) задает координаты нижнего угла квадранта, dX и dY — его размеры. Если размер квадранта еще недостаточно мал, он будет делиться до тех пор, пока значение **Глубина** не станет равным значению **Искомая_Глубина**. Таким образом, исходная область будет разделена на $2^{2*\text{Искомая_Глубина}}$ квадрантов, в каждом из которых будет работать функция **Рисовать_Линию_Уровня**, использующая метод пробных точек. Однако у этого алгоритма есть два существенных недостатка. Во-первых, этот алгоритм может пропустить некоторые маленькие замкнутые контуры или другие ситуации, которые подробно проиллюстрированы на рис. 1. Во-вторых, этот алгоритм весьма неэкономичен, так как он просматривает (сканирует) всю область для рисования контура.

Этих недостатков лишены *интервальные методы*, которые дают абсолютно точный ответ на вопрос, присутствует ли какой-либо участок контура в данной области. Это лучше всего объяснить с помощью рис. 3, на котором показан график некоторой функции f(x) на интервале X=[a,b]. На этом рисунке также показано интервальное расширение F(X)=[A,B] и значения c_1, c_2 и c_3, которые проверяются на принадлежность к интервалу F(X). Так как область изменения функции *всегда* лежит в ее интервальном расширении, следовательно, если значение c_1 не принадлежит интервалу F(X), то оно не может быть и значением функции f(x) на интервале X=[a,b]. Если же некоторое значение принадлежит интервалу F(X), то оно может либо быть значением функции на этом интервале, либо не быть. Как показано на рис. 3, значения c_1 и c_2 таковы, что $c_1 \notin F(X)$ и $c_2 \in F(X)$. В случае двух переменных, если значение линии уровня C не принадлежит интервальному расширению F(X,Y) функции f(x,y) на прямоугольнике $x \in X$, $y \in Y$ плоскости (x,y), эта линия уровня не может существовать на этом прямоугольнике. Именно на этом основан тест на принадлежность контура к данному квадранту, который дает возможность исключать некоторые области из дальнейшего рассмотрения.

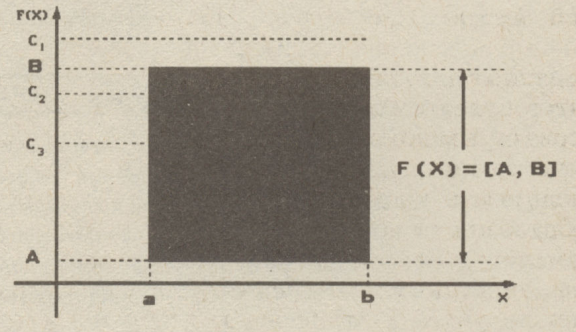


Рис. 3

Эффективность интервальных методов хорошо иллюстрирует рис. 4, на котором изображен контур функции вида

$$\prod_{i=1}^{m} [(X - X_i)^2 (Y - Y_i)^2] \qquad (5)$$

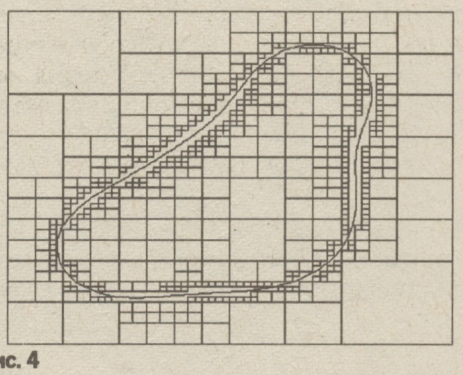


Рис. 4

Уравнение (5) описывает поверхность с минимумами в точках (x_i,y_i). Для успешной работы алгоритму 1, который не использует интервальную арифметику, пришлось бы рассматривать области, в которых отсутствует контур. На рисунке хорошо видно, что в алгоритме, использующем интервальную арифметику, такие области сразу исключаются.

Все сказанное позволяет на основе алгоритма 1 создать очень простой интервальный алгоритм 2, показанный на рис. 5. В этом алгоритме используется не сама функция, а ее интервальное расширение FI. В алгоритме 2 также используется интервальная функция **Есть_Линия_Уровня**, которая возвращает значение *Истина* в случае, если в данном квадранте присутствует искомый контур, то есть $c \in FI(XI, YI)$. Так

```
Процедура Линия_Уровня (Глубина : Глубина_деления;
            X, Y : Точка;
            dX, dY : Расстояние;
            FI : Интервальное_Расширение)

   Процедура Разделить_Квадрант
   {
      Линия_Уровня (Глубина+1, X,      Y,      dX/2, dY/2, FI);
      Линия_Уровня (Глубина+1, X+dX/2, Y,      dX/2, dY/2, FI);
      Линия_Уровня (Глубина+1, X+dX/2, Y+dY/2, dX/2, dY/2, FI);
      Линия_Уровня (Глубина+1, X,      Y+dY/2, dX/2, dY/2, FI);
   }

{
   Если  Глубина < Искомая_Глубина  то
      Если  Есть_Линия_Уровня (X, Y, dX, dY, FI)
         то  Разделить_Квадрант;
   Иначе  Рисовать_Квадрант (X, Y, dX, dY);
}
```

Рис. 5. Алгоритм 2 для рисования линии уровня

как данный алгоритм не использует саму функцию f, контур будет изображаться с помощью процедуры **Рисовать_Квадрант**, которая просто закрашивает другим цветом нужную область. Такой метод можно использовать лишь в том случае, если к разрешению изображения не предъявляются высокие требования. Кроме того, для функций, имеющих примерно одинаковые значения на нескольких соседних интервалах, будет получаться очень грубое изображение линии уровня. Для того чтобы улучшить возможности алгоритма 2, можно предложить следующие дополнения и изменения. Во-первых, в качестве параметра добавить саму функцию f и, во-вторых, процедуру **Рисовать_Квадрант** заменить на процедуру **Рисовать_Линию_Уровня**, которая, как уже говорилось, использует метод пробных точек. Конечно же, не все квадранты, полученные при глубине деления **Искомая_Глубина** будут содержать участки контура, но интервальная функция **Есть_Линия_Уровня** *гарантирует*, что ни один квадрант большего размера не был некорректно исключен из рассмотрения. Окончательно полученный алгоритм 3 приведен на рис. 6.

```
Процедура Линия_Уровня (Глубина : Глубина_деления;
            X, Y : Точка;
            dX, dY : Расстояние;
            F : Функция;
            FI : Интервальное_Расширение)

   Процедура Разделить_Квадрант
   {
      Линия_Уровня (Глубина+1, X,      Y,      dX/2, dY/2, F, FI);
      Линия_Уровня (Глубина+1, X+dX/2, Y,      dX/2, dY/2, F, FI);
      Линия_Уровня (Глубина+1, X+dX/2, Y+dY/2, dX/2, dY/2, F, FI);
      Линия_Уровня (Глубина+1, X,      Y+dY/2, dX/2, dY/2, F, FI);
   }

{
   Если  Глубина < Искомая_Глубина  то
      Если  Есть_Линия_Уровня (X, Y, dX, dY, FI)
         то  Разделить_Квадрант;
   Иначе  Рисовать_Линию_Уровня (X, Y, dX, dY, F);
}
```

Рис. 6. Алгоритм 3 для рисования линии уровня

Построение трехмерных поверхностей

Очевидно, что интервальные методы можно применять не только для двумерных функций. Более того, трехмерные алгоритмы могут быть построены на базе того, что уже описано выше. Из множества подходов и методов построения трехмерных поверхностей можно выделить два, напрямую относящихся к предмету обсуждения.

Первый метод, непосредственно основывающийся на алгоритме 3, позволяет строить линии уровня функции $f(x,y,z)=0$ (то есть заданной в неявном виде) на последовательности плоскостей, параллельных координатным плоскостям. Так как на этих плоскостях одна из переменных x, y или z является константой, такой алгоритм очень легко реализовать. Результат работы такого алгоритма приведен на рис. 7, где показан вид поверхности, задаваемой трехмерным аналогом уравнения (5).

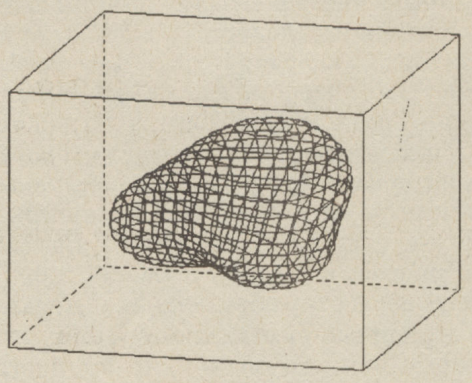


Рис. 7

Можно предложить другой метод, основывающийся на идее деления трехмерного пространства. По своей сути его можно представить как трехмерный аналог алгоритма 3. Для алгоритмизации процесса деления пространства воспользуемся тремя вложенными циклами (в общем, так можно было поступить и в предыдущих алгоритмах). В алгоритме 4 (рис. 8) функция

```
Процедура Поверхность (Глубина : Глубина_деления;
            X, Y, Z : Точка;
            dX, dY, dZ : Расстояние;
            F : Функция;
            FI : Интервальное_Расширение)

   Процедура Разделить_Квадрант
   {
      Для i=0 до 1
         Для j=0 до 1
            Для k=0 до 1
               Поверхность (Глубина+1, X+i*dX/2,
                                       Y+j*dY/2, Z+k*dZ/2, F, FI);
   }

{
   Если  Глубина < Искомая_Глубина  то
      Если  Есть_Поверхность (X, Y, Z, dX, dY, dZ, FI)
         то  Разделить_Квадрант;
   Иначе  Рисовать_Поверхность (X, Y, Z, dX, dY, dZ, F);
}
```

Рис. 8. Алгоритм 4 для рисования поверхности функции, заданной в неявном виде

Есть_Поверхность является трехмерным аналогом функции **Есть_Линия_Уровня**. Функция **Рисовать_Поверхность** вычисляет точки пересечения исследуемой поверхности с некоторым кубом. Графическим результатом может служить полутоно-

вое изображение или триангуляция (аппроксимация треугольниками) соответствующей поверхности. В последнем случае для получения наилучшего результата необходимо применить какой-либо метод удаления невидимых линий.

Практическая реализация интервальных методов

Теперь, когда было дано необходимое теоретическое введение, рассмотрим, как можно практически реализовать описанные методы, и самое главное, применить в них интервальную арифметику. В качестве языка разработки идеально подходит C++ с его перегружаемыми операциями, с помощью которых и будет реализована интервальная арифметика. В строгом соответствии с определением (1) класс *Interval* (листинг 1) состоит из координат границ интервала a и b, арифметических операций и функций, выполняемых с данным интервалом. Исключительно в целях упрощения в классе *Interval* не реализована операция деления на интервал, которая иногда в результате дает объединение двух интервалов. Расширив класс *Interval* указателем на объект того же класса и введя в каждой операции соответствующий цикл, можно реализовать обработку объединения интервалов (однако при этом придется отказаться от inline подстановок, которые немного оптимизируют программу). Введя таким образом интервальную арифметику, можно строить интервальные расширения различных функций, которые выглядят очень естественно, практически не отличаясь по виду от самих функций. Пусть, например, дано каноническое уравнение эллипса, записанное следующим образом:

$$\left(\frac{x}{a}\right)^2 + \left(\frac{y}{b}\right)^2 = 1 \qquad (6)$$

В терминах операций класса *Interval* можно написать соответствующее интервальное расширение в виде чрезвычайно наглядной функции C++:

```
Interval Ellipse_Interval
  (Interval &x, Interval &y)
{ return ((x / a) ^ 2) + ((y / b) ^ 2); }
```

(Оба слагаемых взяты в скобки потому, что операция ^ (в нашем случае — возведение интервала в натуральную степень, а не исключающее ИЛИ) имеет более низкий приоритет, чем операция + (сложение интервалов)). Строго говоря, эта функция описывает лишь левую часть уравнения (6), то есть семейство подобных эллипсов с одним центром в точке (0,0), у которых большая ось относится к малой как a к b. Применяя эту интервальную функцию в алгоритмах 2 и 3 при искомом значении, равном 1, можно получить фигуру, описываемую уравнением (6). На рис. 9 показано семейство таких фигур со смещенным от начала координат центром и с повернутыми на некоторый угол осями.

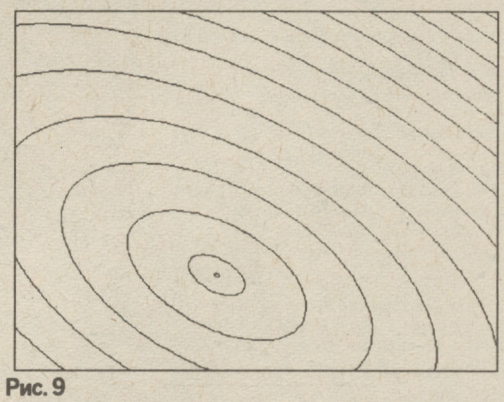


Рис. 9

Для того чтобы применить интервальные методы в программе, потребуется ввести понятия *области пространства* и *точки*. Соответствующий класс *Point* (листинг 2) описывает точку в трехмерном пространстве, храня 3 вещественные координаты. Благодаря этому объекты данного класса можно использовать не только в качестве описания точки с координатами, заданными относительно начала координат, но и в качестве смещения относительно любой другой точки, что будет широко использоваться в дальнейшем. В классе *Point* также определены некоторые операции над точками и их координатами. Операции + и – определяют сложение и вычитание двух векторов, имеющих общее начало в точке (0,0,0) и различные концы с координатами, задаваемыми соответствующими объектами класса *Point*. Операции × и / определяют умножение и деление аналогичного вектора на скаляр. Все эти операции будут широко использоваться при реализации интервального метода.

Как уже упоминалось, нам понадобится понятие *области пространства*, на которой определена исследуемая функция. Понятие трехмерной области пространства описано в классе *Region3D* (листинг 3). Среди функций этого класса — хранение и преобразование вещественных координат в координаты экрана, вычисление различных проекций. Для этого конструктор класса принимает в качестве параметров координаты вещественной области (то есть области изменения функции), координаты экранной области, угол поворота системы координат относительно оси Z и угол проекции осей на плоскость. Обработка этих параметров производится функциями **GetTurnedX** и **GetTurnedY**, которые вычисляют координаты после поворота осей на угол **TurnAngle**, и функциями **GetX** и **GetY**, учитывающими масштаб и угол проекции **ViewAngle** (в работе [3] дано гораздо более подробное описание методов преобразования координат и построения прямоугольных проекций). Предполагается, что координатные оси направлены следующим образом: ось X — вправо, Y — вверх, Z — из плоскости экрана. Нам также понадобится функция **ShowRegion**, которая изображает на экране исследуемую область в виде параллелепипеда, у которого невидимые грани нарисованы пунктиром. Эта функция вызывается каждый раз перед запуском интервально-

го метода. В листинге 3а, где дано описание этого класса, также определены следующие типы:

```
double Function3D(Point&);
Interval Interval3D(   Interval&,
                       Interval&, Interval&).
```

В классе *Region3D* определены функция (указатель) F типа Function3D — ее поверхность мы будем строить — и функция FI типа Interval3D — это интервальное расширение исследуемой функции F.

Остановимся несколько подробнее на остальных функциях этого класса. Функция **ProcessInterval** является практически полным трехмерным аналогом алгоритма 3 (см. предыдущий раздел и рис. 6), за исключением того, что процедура **Разделить_Квадрант** реализована в виде трех вложенных циклов, а функция **Есть_Линия_Уровня** — в виде операции (&&) принадлежности вещественного числа Value интервалу FI в следующем виде:

```
Value && FI(   Interval(...), Interval(...),
               Interval(...))
```

Кроме того, необходимо отметить следующие тонкости. Для того чтобы алгоритм работал корректно, необходимо, чтобы из трех циклов выполнялись только два (не следует забывать, что поверхность мы изображаем в виде совокупности линий уровня, когда одна из трех переменных равна константе). Это достигается при помощи использования в циклах вида

```
for(int i=0; i<=Limit[k]; i++)
```

значений Limit, которые могут быть установлены до вызова данной функции (Limit должны принимать значения 0 или 1 для k-й переменной).

Функции **Рисовать_Линию_Уровня** из алгоритма 3 соответствует функция **Plot3D**, рисующая короткий отрезок контура в малом квадранте с использованием метода пробных точек. Предполагается, что при малых изменениях переменных функция F близка к линейной. Поэтому алгоритм работы **Plot3D** сводится к отысканию точек пересечения контура с границами квадранта и построения прямой между этими точками. Для вычисления знака функции в углах квадранта (рис. 1) и проверки пересечения контура с границами предназначена функция **EdgeCrossed**. Идею, примененную в этой функции, можно проиллюстрировать рисунком 10, на котором показана некоторая двумерная область, определяемая угловыми точками (x, y) и (x+dx, y+dy). Для заданного номера грани (от 0 до 3) определяются начало (Point LLim) и конец грани (Point HLim). Стрелки на рис. 10 указывают направление от LLim к HLim. В этих точках находятся значения функции F, которые сравниваются с искомым значением Value, для которого строится линия уровня. Заметим, что функция **EdgeCrossed** предназначена для исследования функции трех переменных

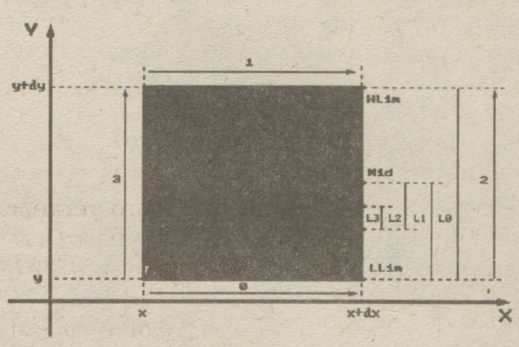


Рис. 10

при условии, что одна из них (любая) не изменяется, то есть одно из значений dx, dy или dz равно нулю. Если знаки на концах грани различны (рис. 1), в функции **Plot3D** применяется метод деления отрезка пополам, являющийся, несмотря на свою простоту, достаточно эффективным. На рис. 10 показано, как применяется этот метод: в середине выбранного отрезка вычисляется значение функции F, из двух полученных отрезков выбирается тот, на котором функция меняет знак, и операция повторяется для одной из половин отрезка до тех пор, пока с заданной точностью не будет найдена точка, где функция обращается в ноль. Такой процесс показан на рис. 10 в виде последовательности отрезков L0...L3, на которых исследуется функция. Между двумя найденными подобным образом точками проводится отрезок прямой. Несмотря на то, что линейная аппроксимация, примененная здесь, является не самым лучшим методом, она дает весьма неплохие результаты: на рис. 11 показаны линии уровня функции, описываемой уравнением (5). Для наименьших значений Value применялось значение SearchDepth = 8, для наибольших — 6.

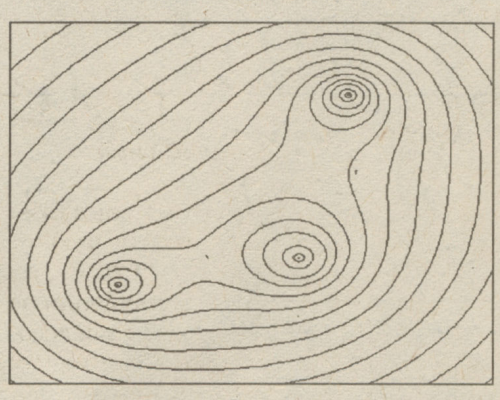


Рис. 11

Кроме описанных функций, в классе *Region3D* содержатся функции **ProcessRegion** и **DrawContour**. **ProcessRegion**, как следует из названия, задает правила работы интервального метода (функции **ProcessInterval**), то есть устанавливает значения Limit и значения переменной, не изменяющей своего значения на каждом шаге. **DrawContour** предназна-

чена для установки переменных и вызова функции **ProcessInterval** внешними функциями. Заметим, что на базе класса *Region3D* можно реализовать также и другие алгоритмы путем наследования, объявив функцию **ProcessInterval** как virtual.

Для того чтобы получить двумерную линию уровня, предлагается класс *Region2D* (листинг 4), который является наследником класса *Region3D*. Конструктор нового класса устанавливает переменную Z, равную некоторой константе, задаваемой в качестве параметра, в остальном он ничем не отличается от конструктора класса — предка. Для преобразования трехмерного пространства в двумерное необходимо переопределить только вычисление координат (функции **GetX** и **GetY**), а также функцию **ShowRegion**, которая теперь будет рисовать прямоугольник соответствующих размеров. Функция **ProcessRegion** также нуждается в переопределении, но в двумерном случае она упрощается до предела: теперь она только устанавливает значения Limit и вызывает функцию **ProcessInterval**, описанную в классе *Region3D*.

В качестве тестового примера в листинге 5 приведена чрезвычайно простая программа, результаты работы которой представлены на рис. 7. Эта программа рисует поверхность, описываемую уравнением — трехмерным аналогом уравнения (5).

Возможные направления дальнейших разработок

Можно предложить несколько путей повышения эффективности приведенных выше методов. Например, для улучшения функций **Рисовать_Линию_Уровня** и **Рисовать_Поверхность** вместо метода пробных точек можно снова применить интервальный метод при отыскании точки пересечения контура с границей квадранта. При практической реализации алгоритма построения трехмерных поверхностей целесообразно сохранять образ поверхности в виде последовательности отрезков, из которых состоят линии уровня, образующие поверхность.

Приведенные разработки могут иметь очень широкое практическое применение. Например, возможность построения поверхности функции, заданной в неявном виде, очень удобна в приложении к аналитической и начертательной геометрии, а также и во многих других областях применения. Например, при помощи программ данной работы очень удобно строить сечения фигур плоскостями: на рис. 12 показана

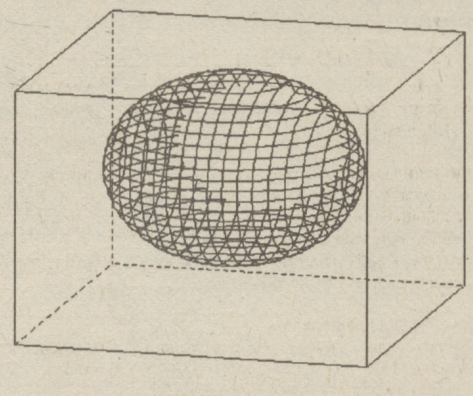


Рис. 12

сфера, «задняя» часть которой отрезана фронтальной плоскостью. Автор надеется, что данная работа послужит хорошей базой для разработок в области компьютерной графики, а также даст пищу для размышления при разработке оригинальных графических алгоритмов. ■

Литература

1. K.G.Suffern, E.D.Fackerell. Interval Methods In Computer Graphics// Comput. & Graphics.– 1991.– № 15.3.
2. R.E.Moore. Interval Analysis.– Prentice-Hall, Englewood Cliffs, 1966.
3. Л. Аммерал. Принципы программирования в машинной графике.– М.: Сол Систем, 1992.

Монитор — 1994 **Подписной купон на 1–е полугодие** МОНИТОР

1	2	3	4

Адрес _____

Ф.И.О. или название организации _____

Стоимость подписки на 1 номер журнала "Монитор" для России — 1000 рублей, для стран СНГ — 1500 рублей. В 1994 году будет выпущено 8 номеров (подписка на полгода — соответственно 4000 руб. и 6000 руб.). Отметьте крестиками те номера журнала, которые Вы хотите получить. Перечислите сумму оплаты (действительна до 31 декабря 1993 г.) на счет НТО "Софт-Москва": р/с 467996 в Независимом Банке России, к/с 1161385 в РКЦ-2 ГУ ЦБ РФ по г. Москве, МФО 201779.
Вышлите подписной купон и квитанцию об оплате по адресу: 117419 Москва, а/я 765.

Листинг 1.
Определение класса Interval — реализация интервальной арифметики

```
#ifndef _INTERVAL_HPP_
#define _INTERVAL_HPP_

#include <math.h>

#define max(a,b) (((a) > (b)) ? (a) : (b))
#define min(a,b) (((a) < (b)) ? (a) : (b))

class Interval
{
public:
   double a, b;                                   // Границы интервала
   inline Interval(double u, double v) {a = u; b = v;} // Конструкторы
   inline Interval(double v) {a = b = v;}

// Вычисление максимума/минимума из четырех чисел
// (используется при умножении интервалов)
   inline friend double min_val(double a, double b, double c, double d)
      { return min( min(a,b), min(c,d)); }
   inline friend double max_val(double a, double b, double c, double d)
      { return max( max(a,b), max(c,d)); }

// Интервальная арифметика:
   // Сложение двух интервалов, интервала и числа, числа и интервала
   inline friend Interval operator +(Interval &x, Interval &y)
      { return Interval(x.a + y.a, x.b + y.b); }
   inline friend Interval operator +(Interval &x, double v)
      { return Interval(x.a + v, x.b + v); }
   inline friend Interval operator +(double v, Interval &y)
      { return Interval(v + y.a, v + y.b); }

// Вычитание одного интервала из другого,
// числа из интервала и наоборот
   inline friend Interval operator -(Interval &x, Interval &y)
      { return Interval(x.a - y.b, x.b - y.a); }
   inline friend Interval operator -(Interval &x, double v)
      { return Interval(x.a - v, x.b - v); }
   inline friend Interval operator -(double v, Interval &y)
      { return Interval(v - y.b, v - y.a); }

// Умножение интервала на интервал, на число и числа на интервал
   inline friend Interval operator *(Interval &x, Interval &y)
      { return Interval( min_val(x.a*y.a, x.a*y.b, x.b*y.a, x.b*y.b),
                         max_val(x.a*y.a, x.a*y.b, x.b*y.a, x.b*y.b)); }
   inline friend Interval operator *(Interval &x, double v)
      { return Interval( min(x.a*v, x.b*v), max(x.a*v, x.b*v)); }
   inline friend Interval operator *(double v, Interval &y)
      { return Interval( min(v*y.a, v*y.b), max(v*y.a, v*y.b)); }

// Деление двух интервалов, интервала на число и числа на интервал
   inline friend Interval operator /(Interval &x, Interval &y)
      { return x*Interval(1/y.b, 1/y.a); }
   inline friend Interval operator /(Interval &x, double v)
      { return x*Interval(1/v, 1/v); }
   inline friend Interval operator /(double v, Interval &y)
      { return v*Interval(1/y.b, 1/y.a); }

// Некоторые математические функции, определенные на интервалах
   inline friend Interval exp(Interval &x) // Экспонента
      { return Interval(exp(x.a), exp(x.b)); }
   inline friend Interval ln(Interval &x) // Натуральный логарифм
      { return Interval(log(x.a), log(x.b)); }
   inline friend Interval sqrt(Interval &x) // Квадратный корень
      { return Interval(sqrt(x.a), sqrt(x.b)); }

// Возведение интервала в натуральную степень
   inline friend Interval operator ^(Interval &x, int k)
      { if(k < = 1) return x; else return x*(x^(k-1)); }

// Логическая операция принадлежности точки интервалу
   inline friend int operator &&(double v, Interval &x)
      { if(x.a < = v && v < = x.b) return 1; else return 0; }
};
#endif
```

Листинг 2a.
Определение класса POINT — точка в трехмерном пространстве

```
#ifndef _POINT_HPP_
#define _POINT_HPP_

class Point
{
public:
   double x, y, z;                          // Координаты точки
   Point(double, double, double);           // Конструкторы
   Point(double, double);
   Point();

   // Некоторые действия над точками и их координатами
   friend Point operator+(Point&, Point&);
   friend Point operator-(Point&, Point&);
   friend Point operator*(Point&, double);
   friend Point operator/(Point&, double);
   // Сравнение координат двух точек
   friend int operator < = (Point&, Point&);
};
#endif
```

Листинг 2b.
Класс Point — точка в трехмерном пространстве

```
#include "Point.hpp"

//::::::::::::::::::::::::::::::::::::::::::::: Конструкторы ::::::::::::::::::::::::::::::::::::::::::::
Point::Point(double a, double b, double c) { x = a; y = b; z = c;}
Point::Point(double a, double b) { x = a; y = b; z = 0; }
Point::Point() { x = y = z = 0; }

//:::::::::::::::::::::::::: Операции над точками и их координатами ::::::::::::::::::
Point operator +(Point &a, Point &b)
{ return Point(a.x+b.x, a.y+b.y, a.z+b.z); }

Point operator -(Point &a, Point &b)
{ return Point(a.x-b.x, a.y-b.y, a.z-b.z); }

Point operator *(Point &a, double c)
{ return Point(a.x*c, a.y*c, a.z*c); }

Point operator /(Point &a, double c)
{ return Point(a.x/c, a.y/c, a.z/c); }

// Сравнение координат точек
int operator < = (Point &a, Point &b)
{ return (a.x < = b.x && a.y < = b.y && a.z < = b.z); }
```

Листинг 3a.
Определение класса Region3D — область трехмерного пространства

```
#ifndef _REGION3D_HPP_
#define _REGION3D_HPP_
#include "Point.hpp"
#include "Interval.hpp"

// Тип исследуемой функции трех переменных
typedef double Function3D(Point&);

// Тип интервального расширения Function3D
typedef Interval Interval3D(Interval&, Interval&, Interval&);

class Region3D
{
private:
   double TurnAngle;                // Угол поворота осей
   double ViewAngle;                // Угол проекции
   double SinTA, CosTA, SinVA;      // Соответствующие синусы и косинусы

   double GetTurnedX(Point&);       // Вычисление преобразования
   double GetTurnedY(Point&);       // поворота

protected:
   double ScaleX, ScaleY, ScaleZ;   // Масштабы координатных осей
   Point *C1, *C2;                  // Границы области
   Point *ScrC1, *ScrC2;            // Экранные соответствия границ области

   double Value;            // Отыскиваемое значение функции
   int SearchDepth;         // Искомая глубина поиска
```

```
    int ShowQuadrant;       // Следует ли рисовать подобласти
    Point LLim, HLim;       // Эти значения используются в функции Plot3D
    Function3D *F;          // Функция, линия уровня которой строится
    Interval3D *FI;         // Ее интервальное расширение
    int Limit[3];           // Задание изменяемых переменных

    int Round(double x);        // Функция округления координат
    virtual int GetX(Point&);   // Вычисление координат по осям
    virtual int GetY(Point&);

    // Рисование маленького отрезка линии уровня .
    void Plot3D(Point &C1, Point &C2);

    // Проверка на пересечение заданного отрезка линией уровня
    int EdgeCrossed(int Count, Point &A, Point &dA);

    // Рекурсивная функция исследования интервала (собственно
    // реализация интервального метода
    virtual void ProcessInterval(Point &C1, Point &C2, int Depth);
    virtual void ProcessRegion();

public:
    virtual void ShowRegion(Point&, Point&);

    Region3D(Point *C1, Point *C2, Point *ScrC1, Point *ScrC2,
             int turn_angle, int view_angle);
    void DrawContour(double value, int search_depth,
             Function3D, Interval3D, int show_quadrant);
};
#endif
```

Листинг 3b. Класс Region3D — область трехмерного пространства

```
#include "Region3D.hpp"
#include <graphics.h>
#include <math.h>
//::::::::::::::::::::::::::::::::::::::::::::: Конструктор ::::::::::::::::::::::::::::::::::::::::::::
Region3D::Region3D(Point *c1, Point *c2, Point *scrc1, Point *scrc2,
          int turn_angle, int view_angle) {
    ScrC1 = scrc1;          // Экранные соответствия координат
    ScrC2 = scrc2;
    C1 = c1; C2 = c2;       // Границы области

    // Вычисление масштабов
    ScaleX = (ScrC2->x - ScrC1->x)/(C2->x - C1->x);
    ScaleY = (ScrC2->y - ScrC1->y)/(C2->y - C1->y);
    ScaleZ = (ScrC2->z - ScrC1->z)/(C2->z - C1->z);

    TurnAngle =  M_PI/180 * turn_angle;          // Перевод из градусов
    ViewAngle =  M_PI/180 * view_angle;          // в радианы

    CosTA = cos(TurnAngle); SinTA = sin(TurnAngle);
    SinVA = sin(ViewAngle);

    Limit[0] = Limit[1] = 1; Limit[2] = 0;
} //Region3D
//::::::::::::::::::::::::::::::::::::::::::::: Функции-члены ::::::::::::::::::::::::::::::::::::::::::::
//.....................................................Round.....................................................
int Region3D::Round(double x) { // Округление числа x
    double Integer;    // Хранение целой части числа
    return (int) (modf(x, &Integer) < 0.5 ? Integer : Integer+1);
} // Round
//.....................................................GetTurned*.....................................................
double Region3D::GetTurnedX(Point &A) // Поворот системы координат
{ return (A.x - C1->x)*CosTA - (A.y - C1->y)*SinTA - C1->x + C2->x; }

double Region3D::GetTurnedY(Point &A)
{ return (A.x - C1->x)*SinTA + (A.y - C1->y)*CosTA - C1->y + C2->y; }

//.....................................................Get*.....................................................
int Region3D::GetX(Point &A)          // Масштабирование
{ return Round(GetTurnedX(A)*ScaleX + ScrC1->x); }

int Region3D::GetY(Point &A)          // и вычисление проекции
{ return Round(GetTurnedY(A)*ScaleY*SinVA + ScrC1->y -
        (A.z - C1->z + C2->z)*ScaleZ); }

//.....................................................ShowRegion.....................................................
// Функция показывает границы трехмерной области
void Region3D::ShowRegion(Point &A, Point &dA) {
    int Polygon[6];                   // Углы прямоугольника
    Point Corner[4];                  // то же самое, но в виде Point
    // Рисуем горизонтальные грани
    for(int i = 0; i <= 1; i++) {          //Вычислим координаты углов
        Corner[0] = Point(A.x,          A.y,          A.z+i*dA.z);
        Corner[1] = Point(A.x+dA.x,  A.y,          A.z+i*dA.z);
        Corner[2] = Point(A.x+dA.x,  A.y+dA.y,  A.z+i*dA.z);
        Corner[3] = Point(A.x,          A.y+dA.y,  A.z+i*dA.z);

        // Рисуем "задние" грани
        // Занесем координаты углов в массив
        Polygon[0] = GetX(Corner[3]); Polygon[1] = GetY(Corner[3]);
        Polygon[2] = GetX(Corner[0]); Polygon[3] = GetY(Corner[0]);
        Polygon[4] = GetX(Corner[1]); Polygon[5] = GetY(Corner[1]);

        // Невидимые линии будем рисовать пунктиром
        int LineStyle =  (i == 0) ? DASHED_LINE : SOLID_LINE;
        setlinestyle(LineStyle, 1, 1);
        drawpoly(3, Polygon);

        // Рисуем "передние" грани
        // Занесем остальные точки в массив
        Polygon[0] = GetX(Corner[1]); Polygon[1] = GetY(Corner[1]);
        Polygon[2] = GetX(Corner[2]); Polygon[3] = GetY(Corner[2]);
        Polygon[4] = GetX(Corner[3]); Polygon[5] = GetY(Corner[3]);

        setlinestyle(SOLID_LINE, 1, 1);
        drawpoly(3, Polygon);
    } // for

    for(i = 0; i<4; i++) {                 // Рисуем вертикальные грани
        int LineStyle =  (i == 0) ? DASHED_LINE : SOLID_LINE;
        setlinestyle(LineStyle, 1, 1);
        line(GetX(Corner[i]), GetY(Corner[i]), GetX(Corner[i]),
             GetY(Point(Corner[i].x, Corner[i].y, Corner[i].z - dA.z)));
    } // for
} // ShowRegion

//.....................................................ProcessRegion.....................................................
void Region3D::ProcessRegion() {
    // Вычислим размер области как координаты точки
    Point Size = Point(C2->x - C1->x, C2->y - C1->y, C2->z - C1->z);
    ShowRegion(*C1, Size);                 // Покажем всю область

    for(int k = 0; k <= 1; k++) {
        switch(k) {
            case 0: Limit[0] = Limit[1] = 1; Limit[2] = 0; break;
            case 1: Limit[1] = Limit[2] = 1; Limit[0] = 0; break;
        } // switch
        Point Plane2 = Point((C2->x - C1->x)*Limit[0],
                             (C2->y - C1->y)*Limit[1],
                             (C2->z - C1->z)*Limit[2]);
        for(int iMax = 25, i = 0; i <= iMax; i++) {
            Point Plane1 = Point(C1->x + (C2->x-C1->x)*i/iMax*(!Limit[0]),
                                 C1->y + (C2->y-C1->y)*i/iMax*(!Limit[1]),
                                 C1->z + (C2->z-C1->z)*i/iMax*(!Limit[2]));
            ProcessInterval(Plane1, Plane2, 0);        // Вызов метода
        } // for i
    } // for k
} // ProcessRegion

//.....................................................EdgeCrossed.....................................................
// Функция реализует метод пробных точек в двумерном пространстве,
// подробные объяснения см. в тексте
int Region3D::EdgeCrossed(int NumEdge, Point &A, Point &dA) {
    switch(NumEdge) {                      // Зададим порядок обхода граней
        case 0:  LLim = Point(A.x,          A.y,          A.z);
                 HLim = Point(A.x + dA.x,  A.y,          A.z + dA.z);  break;
        case 1:  LLim = Point(A.x + dA.x,  A.y,          A.z);
                 HLim = Point(A.x + dA.x,  A.y + dA.y,  A.z);          break;
        case 2:  LLim = Point(A.x,          A.y + dA.y,  A.z);
                 HLim = Point(A.x + dA.x,  A.y + dA.y,  A.z + dA.z);  break;
        case 3:  LLim = Point(A.x,          A.y,          A.z + dA.z);
                 HLim = Point(A.x,          A.y + dA.y,  A.z + dA.z);  break;
    } // switch

    // Значения функции на концах отрезка
    double FLow = F(LLim), FHigh = F(HLim);
    // Проверка знаков на концах отрезка
    if ((FLow - Value) * (FHigh - Value) >= 0) return 0;// Знаки одинаковы
    if (Value >= 0)   return (FHigh > FLow ? 1 : -1);     // Разные знаки
    else return (FHigh < FLow ? 1 : -1);
} // EdgeCrossed

//.....................................................Plot3D.....................................................
// Функция рисования маленького отрезка линии
void Region3D::Plot3D(Point &A, Point &dA) {
    double Delta = fabs(dA.x*dA.x+dA.y*dA.y)/100;
```

```
    for(int i = 0, Edges = 0; i < 4; i++) { // Обойдем все 4 стороны квадранта
      int Sign = EdgeCrossed(i, A, dA); // Проверим знаки
      if(Sign! = 0) {                   // Обнаружено пересечение
        while(LLim < = HLim) {   // Реализация метода деления отрезка
          Point Mid = (HLim+LLim)/2;        // Середина отрезка
          double MidF = F(Mid);
          if( fabs(fabs(MidF) - fabs(Value)) < = Delta) {
            if(Edges = = 0) {           // Найдена первая точка
              moveto(GetX(Mid), GetY(Mid));
              Edges++;
            }                           // Найдена вторая точка
            else lineto(GetX(Mid), GetY(Mid));
            break;                      // Закончить вычисления
          } // if
          if((fabs(MidF) - fabs(Value))*Sign > 0 ) HLim = Mid;
          else LLim = Mid;
        } // while
      } // if
    } // for
  } // Plot3D

  //..... ProcessInterval .....
  // Собственно реализация интервального метода
  void Region3D::ProcessInterval(Point &A, Point &dA, int Depth) {
    if(Depth < SearchDepth) { // Если не достигнута нужная глубина деления
      for(int i = 0; i < = Limit[0]; i++) {     // Цикл дает возможность
        for(int j = 0; j < = Limit[1]; j++) {   // задать 4 точки
          for(int k = 0; k < = Limit[2]; k++) {
            double   ix = A.x + i*dA.x / 2,
                     iy = A.y + j*dA.y / 2,
                     iz = A.z + k*dA.z / 2;
            Point NewA = Point(ix, iy, iz); // Координаты нового квадранта
            Point NewdA = dA/2;    //Point(dA.x/2, dA.y/2, dA.z/2);
            if( Value && FI(  Interval(ix, ix+dA.x/2*Limit[0]),
                              Interval(iy, iy+dA.y/2*Limit[1]),
                              Interval(iz, iz+dA.z/2*Limit[2])))
              ProcessInterval(NewA, NewdA, Depth+1);
            else if(ShowQuadrant) ShowRegion(NewA, NewdA);
          } // for k
        } // for j
      } // for i
    } // if
    else Plot3D(A, dA);                     // Изобразим линию
  } // ProcessInterval

  //..... DrawContour .....
  // Функция вызова интервального метода
  void Region3D::DrawContour(double value, int search_depth,
       Function3D f, Interval3D fi, int show_quadrant) {
    Value = value;                          // Установка значений переменных
    SearchDepth = search_depth;
    ShowQuadrant = show_quadrant;
    F = f; FI = fi;
    ProcessRegion();
  } // DrawContour
```

■

Листинг 4a.
Определение класса Region2D — область двумерного пространства, на которой определена исследуемая функция

```
#ifndef _REGION2D_HPP_
#define _REGION2D_HPP_
#include "Region3D.hpp"

class Region2D: public Region3D
{
private:
    double ConstZ;        // Константа, подставляемая вместо переменной z
    int GetX(Point&);     // Вычисление координат по осям
    int GetY(Point&);
    void ShowRegion(Point&, Point&); // Изображение границ области
    void ProcessRegion();            // Запуск интервального метода
public:
    // Конструктор
    Region2D(Point *C1, Point *C2, Point *ScrC1, Point *ScrC2,
             double const_z);
};
#endif
```

■

Листинг 4b.
Класс Region2D — область двумерного пространства, на которой определена исследуемая функция

```
#include "Region2D.hpp"
#include <graphics.h>

//:::::::::::::::::::::::::::::::::::::::: Конструктор ::::::::::::::::::::::::::::::::::::::::
Region2D::Region2D(Point *C1, Point *C2,
          Point *ScrC1, Point *ScrC2, double const_z):
               Region3D(C1,C2, ScrC1,ScrC2, 0,0) {
  ConstZ = const_z;
} // Region2D

//:::::::::::::::::::::::::::::::::::::::: Функции-члены ::::::::::::::::::::::::::::::::::::::::

//..... Get* .....
int Region2D::GetX(Point &A)            // Вычисление координаты x
{ return Round((A.x - C1->x)*ScaleX + ScrC1->x); }

int Region2D::GetY(Point &A)            // Вычисление координаты y
{ return Round(ScrC2->y - (A.y - C1->y)*ScaleY + ScrC1->y); }

//..... ShowRegion .....
// Функция показывает границы области
void Region2D::ShowRegion(Point &A, Point &dA) {
  rectangle(GetX(A), GetY(A), GetX(A+dA), GetY(A+dA));
} // ShowRegion

//.....ProcessRegion.....
// Функция запускает интервальный метод
void Region2D::ProcessRegion()
{
  // Вычислим размер области как координаты точки
  Point Size = Point(C2->x - C1->x, C2->y - C1->y, ConstZ);
  C1->z = ConstZ;  // Эта координата будет константой
  ShowRegion(*C1, Size);                // Покажем всю область
  ProcessInterval(*C1, Size, 0);        // Вызов метода
} // ProcessRegion
```

■

Листинг 5.
Тестовая программа

```
#include "Region3D.hpp"
#include <math.h>

// Исследуемая функция (трехмерный аналог уравнения (5))
double F(Point &A)
{ return ( pow(A.x+1.1,2) + pow(A.y+0.9,2) )*
         ( pow(A.x-0.8,2) + pow(A.y-1.2,2) )*
         ( pow(A.x-0.4,2) + pow(A.y+0.6,2) ); }
// Ее интервальное расширение
Interval FI(Interval &x, Interval &y, Interval &z)
{ return ( ((x+1.1)^2) + ((y+0.9)^2) )*
         ( ((x-0.8)^2) + ((y-1.2)^2) )*
         ( ((x-0.4)^2) + ((y+0.6)^2) ); }
// Вещественные границы области
Point Point1 = Point(-2, -2, -2), Point2 = Point(2, 2, 2);
// Экранные границы области
Point Screen1 = Point(-100,290,100), Screen2 = Point(250,500,350);
int TurnAngle = 15, ViewAngle = 24;     // Углы поворота и проекции
int SearchDepth = 5, ShowQuadrant = 0; // Глубина деления пространства
double Contour = 15;              // Искомое значение функции (поверхности)

int main(void)
{
  if(InitGraphics() = = 0) return 1;    // Инициализируем графику

  // Определим объект класса Region3D
  Region3D MyRegion3D = Region3D(&Point1, &Point2, &Screen1,
                                 &Screen2, TurnAngle, ViewAngle);
  // Изобразим поверхность функции F
  MyRegion3D.DrawContour(Contour, SearchDepth, F, FI, ShowQuadrant);

  CloseGraphics();                      // Закроем графику
  return 0;
}
```

■

Как масштабировать растровые картинки

А. Крупник

Очень часто нас не удовлетворяет размер растровой картинки. Хочется, чтобы она была больше или меньше. Но как добиться этого? Ясно, что здесь возможностей гораздо меньше, чем в случае картинок, представленных в векторном виде, то есть в виде набора прямых, окружностей и так далее. И тем не менее задача масштабирования растровых картинок вполне разрешима. Многие графические программы, такие как IMAGE 72, IMAGE ALCHEMY или PHOTOSTYLER, с этой задачей справляются. Попробуем и мы этому научиться.

Начнем, как обычно, с основной идеи, которая, с одной стороны, неочевидна, с другой — поражает своей примитивностью, с третьей — не так плоха, как это может показаться. Итак, пусть у нас есть растровая картинка, в которой 8 пикселов по горизонтали и 14 — по вертикали, то есть в картинке 14 строчек и 8 столбцов. Допустим, нам нужна картинка, в которой 20 строчек и 8 столбцов. Откуда взять лишние 6 строчек? Конечно же, из исходной картинки. Пронумеруем строчки в исходной картинке от 1 до 14. Тогда в масштабированной картинке эти строчки будут идти в таком порядке: 1, 2, 2, 3, 4, 4, 5, 6, 6, 7, 8, 9, 9, 10, 11, 11, 12, 13, 13, 14. Иными словами, для того чтобы наскрести еще 6 строчек, приходится дублировать строчки 2, 4, 6, 9, 11, 13. А как быть в том случае, когда исходную картинку нужно уменьшить? Ясно, что не все строчки из исходной картинки будут использованы. Если, например, исходное изображение, в котором 14 строчек, нужно превратить в картинку с семью строчками, то следует из исходного изображения взять каждую вторую строчку, то есть строки с номерами 1, 3, 5, 7, 9, 11, 13 или четные строки с номерами 2, 4, 6, 8, 10, 12, 14. Последний пример наглядно показывает, что задача масштабирования растровой картинки не имеет единственного решения.

Обратимся теперь к подпрограмме на ассемблере, которая осуществляет масштабирование черно-белых картинок в графическом редакторе IMAGE 72 (листинг 1). Для понимания программы нужно прежде всего знать, в каком формате представлены растровые картинки. Этот формат крайне прост (в IMAGE 72 изображения, записанные в этом формате, имеют расширение PUT). В первых двух байтах (слове) записана ширина картинки в пикселах, второе слово содержит высоту картинки. Далее расположен бинарный образ изображения, которое кодируется по принципу один пиксел — один бит. Единичный бит соответствует белому цвету, нулевой — черному. Строка изображения представлена целым числом байтов. Если ширина картинки не кратна 8, то лишние биты обнуляются. На рисунке 1 представлено изображение, в котором 11 строк и 11 столбцов, и шестнадцатеричный дамп этого изображения, записанного в формате .PUT.

Рис. 1. Черно-белое изображение 11×11 и его запись в формате .PUT

Теперь работу подпрограммы можно описать более подробно. На словах это выглядит примерно так.

1. Берется строка из исходной картинки и определяется, сколько раз она будет продублирована. Если ноль — строка пропускается. Переходим к п. 7.
2. Если строка не пропускается, то из нее берется пиксел и определяется, сколько раз он будет продублирован. Если ноль, пиксел пропускается.
3. Если не ноль, пиксел дублируется необходимое число раз.
4. Берется следующий пиксел из исходной строки и вновь определяется, дублировать его несколько раз или пропустить.

5. Пункт 4 повторяется до тех пор, пока не будет просканирована вся исходная строка. После этого в новом изображении появится масштабированная строка.
6. Масштабированная строка дублируется. Количество дублей определено в п. 1.
7. Переходим к следующей строке в исходной картинке и возвращаемся к п. 1.

Действия, описанные в пунктах 1-7, повторяются до тех пор, пока не будут исчерпаны все строки в исходной картинке.

Те, кто дочитал статью до этого места, неизбежно спросят: «Как определить, дублировать данную строку или пропускать? И если дублировать, то сколько раз?»

Мы подошли к самому, на мой взгляд, интересному и трудному месту в описываемом алгоритме. Чтобы понять прием, позволяющий вставлять строки в новое изображение или удалять строки из старого, рассмотрим пример 1, представляющий собой фрагмент подпрограммы, ответственный за масштабирование по вертикали. В строке 5 высота новой картинки делится на высоту старой. При этом в регистре AX находится частное, а в регистре DX — остаток. В строке 6 частное запоминается в слове **HightQuot**. Далее, в строке 7 регистр AX обнуляется и в строке 8 снова производится деление на высоту старой картинки. Каков смысл этой операции? Так как в регистре DX был остаток, то в слове **HightRemnd** будет запомнено частное от деления этого остатка, умноженного на два в шестнадцатой степени, на высоту старой картинки. Эти две операции деления лучше представить в виде равенств:

```
HDst = HSrc * HightQuot + Remainder                (1)
Remainder * 10000h =
  = HSrc * HightRemnd + NewRemainder               (2)
```

Заметим, что пара слов **HightQuot**, **HightRemnd** представляет собой запись отношения **HDst/HSrc** в виде числа с фиксированной точкой. **HightQuot** — это целая часть, а **HightRemnd** — дробная. Значению **HightRemnd**=8000h соответствует число 1/2=0,5, значению 0C000h — 1/2+1/4=0,75 и так далее. Теперь осталось самое сложное — понять смысл того, что делается в строках 12 и 13. В строке 12 к содержимому BX прибавляется **HightRemnd**, а в строке 13 суммируется флаг переноса и **HightQuot**. Когда выполнение строки 12 приведет к установке CF (флага переноса)? Очевидно, когда результат сложения будет больше или равен 10000h. А что будет после того, как произойдет перенос? Ясно, что в строке 13 CX станет на единицу больше, чем обычно. Значит, установка CF сигнализирует о том, что будет вставлена лишняя строчка, и остается лишь убедиться в том, что будет вставлено количество строк, равное **Remainder**.

Простые рассуждения покажут нам, что это действительно так. Представим себе, что BX=0. Будем прибавлять к регистру BX по единице. Ничего странного не будет до того момента, когда BX станет равен 0ffffh. Если теперь прибавить еще единицу, регистр обнулится и будет установлен CF. Значит, при суммировании единиц CF будет устанавливаться столько раз, сколько раз в сумме содержится число 10000H. А что изменится, если в регистре BX будут суммироваться другие числа, скажем, пятерки? Да ничего. Ведь прибавить пятерку — это все равно, что пять раз прибавить единицу.

ГРАФИКА

Основная идея алгоритма,
с одной стороны, неочевидна,
с другой — поражает своей
примитивностью,
с третьей — не так плоха,
как может показаться.

Теперь вернемся к нашему алгоритму и еще раз посмотрим на пример 1. Так как при масштабировании исследуются все строки старого изображения, сложение в строке 12 будет произведено **HSrc** раз. Посмотрим на равенство (2), из которого следует, что суммирование **HightRemnd HSrc** раз даст число

```
Remainder * 10000h - NewRemainder
```

Если в начале суммирования BX=**NewRemainder**=0, CF установится **Remainder** раз. Но если **NewRemainder** отлично от нуля, приходится страховаться и засылать в BX некое число, чтобы не потерять одну установку CF. В нашем случае в BX засылается число 8000h=32768d. Значит, чтобы не потерять установку CF, нужно, чтобы **NewRemainder** было меньше 8000h. Но из равенства (2) видно, что **NewRemainder** < **HSrc**. Отсюда следует, что на **HSrc** накладывается ограничение: высота исходной картинки не должна быть больше 32768, что для большинства случаев вполне достаточно!

В листинге 2 представлена тестовая программа, иллюстрирующая работу алгоритма масштабирования. Программа высвечивает на экране букву «S» и ее увеличенную копию. Чтобы выйти из программы, нажмите клавишу ESC.

Описанный только что алгоритм оставляет нехорошее чувство. Дублирование или исключение строчек — путь слишком прямолинейный и явно не лучший. Хотите доказательств — поработайте с пакетом **PHOTOSTYLER**! Как устроено масштабирование в этом пакете — загадка. Может быть, кто-то сумеет ее разгадать, и тогда появится еще одна статья об этом увлекательном и загадочном масштабировании.

В заключение хочу поблагодарить С.В.Беляева за то, что он прочитал статью и сделал ряд ценных замечаний. ■

Пример 1.
Масштабирование по вертикали.
Фрагмент алгоритма

```
             mov     bp,[si+02]          ; 1 Прочитать HSrc
             mov     HSrc,bp             ; 2
             mov     ax,es:[di+02]       ; 3 Прочитать HDst
             xor     dx,dx               ; 4
             div     bp                  ; 5 Поделить HDst на HSrc
             mov     HightQuot,ax        ; 6 Целая часть HDst/HSrc
             xor     ax,ax               ; 7 Остаток, умноженный на 10000h
             div     bp                  ; 8 Поделить на HSrc
             mov     HightRemnd,ax       ; 9 Дробная часть HDst/HSrc
             mov     bx,8000h            ; 10 Заслать в сумматор 1/2
ScaleIni:    xor     cx,cx               ; 11
             add     bx,cs:HightRemnd    ; 12 Если CF = 1, вставить одну строку
             adc     cx,cs:HightQuot     ; 13 Дублировать Src-строку cx раз
             push    bx                  ; 14
             jz      NxtSrcLn            ; 15 Если ноль, пропустить строку

NxtSrcLn:    pop     bx

             dec     cs:HSrc             ; Все строки исследованы?
             jnz     ScaleIni            ; Нет - перейти к новой Src-строке
             ret                         ; Да - конец масштабирования
Scale        endp
```

Листинг 1.
Подпрограмма масштабирования SCALE.INC

```
; На входе:
; DS:SI указывает на исходную .PUT картинку
; ES:DI указывает на преобразованную картинку
; ES:[DI+00] - новая ширина
; ES:[DI+02] - новая высота
; Условные обозначения в комментариях:
; WSrc - ширина исходной картинки
; HSrc - высота исходной картинки
; WDst - ширина новой картинки
; HDst - высота новой картинки
Scale proc near
        mov  bx,[si]          ; Прочитать WSrc
        mov  bp,[si+02]       ; Прочитать HSrc
        add  si,04h           ;
        mov  ax,bx            ;
        add  ax,0007h         ;
        shr  ax,1             ; Поделить
        shr  ax,1             ; на
        shr  ax,1             ; 8
        mov  BplSrc,ax        ; Запомнить число
                              ; байтов в строке
        push ds               ;
        push cs               ;
        pop  ds               ; Восстановить DS
        mov  HSrc,bp          ;
        mov  WSrc,bx          ;
        mov  ax,es:[di]       ; Прочитать WDst
        xor  dx,dx            ;
        div  bx               ; Поделить на WSrc
        mov  WideQuot,ax      ; Запомнить целую
                              ; часть WDst/WSrc
        xor  ax,ax            ; Остаток, умноженный
                              ; на 10000h
        div  bx               ; Поделить на WSrc
        mov  WideRemnd,ax     ; Запомнить дробную
                              ; часть WDst/WSrc
        mov  ax,es:[di+02]    ; Прочитать HDst
        xor  dx,dx            ;
        div  bp               ; Поделить HDst на HSrc
        mov  HightQuot,ax     ; Запомнить целую
                              ; часть HDst/HSrc
        xor  ax,ax            ; Остаток, умноженный
                              ; на 10000h
        div  bp               ; Поделить на HSrc
        mov  HightRemnd,ax    ; Запомнить дробную
                              ; часть HDst/HSrc
        mov  bx,8000h         ; Заслать в сумматор 1/2
        mov  ax,bx            ; ah - битовая маска
                              ; для Src-байта
        add  di,04            ;
        pop  ds               ;
ScaleIni:
        push ax               ; Начало масштабирования
        push si               ;
        xor  cx,cx            ;
        add  bx,cs:HightRemnd ; Если CF=1
                              ; вставить одну строку
        adc  cx,cs:HightQuot  ; Дублировать
                              ; Src-строку cx раз
        push bx               ;
        jz   NxtSrcLn         ; Если ноль -
                              ; пропустить строку
        push cx               ;
        push di               ;
        mov  al,00            ;
        mov  bx,cs:WideRemnd  ; В bx - дробная
                              ; часть WDst/WSrc
        mov  dl,80h           ; dl - битовая маска
                              ; для Dst-байта
        mov  bp,8000h         ; Заслать в сумматор 1/2
        mov  cx,cs:WSrc       ;
LineScale:
        push cx               ; Перебор пикселов
                              ; в Src-строке
        xor  cx,cx            ;
        add  bp,bx            ; Если CF=1, вставить
                              ; один пиксел
        adc  cx,cs:WideQuot   ; Дублировать
                              ; Src-пиксел cx раз
        jz   NxtSrcPix        ; Ноль -
                              ; пропустить пиксел
        mov  dh,[si]          ; Прочитать Src-байт
NextPixel:
        test ah,dh            ; Единичный пиксел?
        jz   SkipPix          ; Нет - пропустить
        or   al,dl            ; Да - записать
SkipPix:
        ror  dl,01            ; Указатель - на
                              ; следующий Dst-пиксел
        jnb  NxtLoop          ; Dst-байт не заполнен?
                              ; Продолжить
        stosb                 ; Dst-байт заполнен?
                              ; Записать
        xor  al,al            ; Очистить байт
NxtLoop:
        loop NextPixel        ;
NxtSrcPix:
        ror  ah,1             ; Указатель - на
                              ; следующий Src-пиксел
        jnb  SameByte         ; Src-байт просмотрен?
        inc  si               ; Да - переместиться
                              ; на следующий байт
SameByte:
        pop  cx               ; Нет - исследовать
                              ; тот же байт
        loop LineScale        ; Исследовать
                              ; следующий пиксел
        or   dl,dl            ; Один байт не записан?
        js   DstLnCopy        ; Нет - дублировать
                              ; Dst-строку
        stosb                 ; Да - записать
                              ; оставшийся байт
DstLnCopy:
        pop  si               ; si указывает на
                              ; начало Dst-строки
        pop  bp               ; Строку нужно
                              ; копировать bp раз
        mov  cx,di            ; di указывает на
                              ; конец Dst-строки
        sub  cx,si            ; В Dst-строке cx байт
        push ds               ;
        push es               ;
        pop  ds               ;
CopyLine:
        dec  bp               ; Уменьшить счетчик
                              ; строк
        jz   CopyOk           ; Все строки
                              ; скопированы?
        push cx               ;
        repz movsb            ; Нет - дублировать
                              ; Dst-строку
        pop  cx               ;
        sub  si,cx            ; Указатель - на
                              ; начало строки
        jmp  short CopyLine   ; Копировать еще раз
CopyOk:
        pop  ds               ;
NxtSrcLn:
        pop  bx               ;
        pop  si               ;
        pop  ax               ;
        add  si,cs:BplSrc     ; Указатель - на
                              ; следующую Src-строку
        dec  cs:HSrc          ; Все строки
                              ; исследованы?
        jnz  ScaleIni         ; Нет - перейти
                              ; к новой Src-строке
        ret                   ; Да - конец
                              ; масштабирования

BplSrc dw ?
        ; Число байтов в строке Src-картинки
HSrc   dw ?
        ; Ширина Src-картинки в пикселах
WSrc   dw ?
        ; Высота Src-картинки в пикселах
WideQuot   dw ?
        ; Целая часть WDst/WSrc
WideRemnd  dw ?
        ; Дробная часть WDst/WSrc
HightQuot  dw ?
        ; Целая часть HDst/HSrc
HightRemnd dw ?
        ; Дробная часть HDst/HSrc
Scale endp
```

Листинг 2.
Тестовая программа TEST.ASM

```
; tasm /m2 test.asm
; tlink /t test.obj
; Высвечивает на экране исходную картинку
; и ее масштабированную копию
; Необходим видеоадаптер EGA
.radix     16
cseg       segment
assume     cs:cseg
           org  0100h
SclTest    proc far
           mov  ah,0fh         ; Получить текущие
```

```
                        ; видеорежим
        int  10h        ; и видеостраницу
        mov  old_v_mode,al ; Запомнить видеорежим
        mov  old_d_page,bh ; Запомнить
                        ; видеостраницу
        call SetVMode   ; Установка режима EGA
                        ; 640x350
        mov  si,OFFSET SrcPict
                        ; Получить смещения
                        ; исходной
        mov  di,OFFSET DstPict
                        ; и масштабированной
                        ; картинки
        call Scale      ; Масштабировать
        mov  si,OFFSET SrcPict ;
        mov  cx,00      ; Горизонтальная
                        ; координата
        mov  dx,00      ; Вертикальная
                        ; координата
        call PicDisp    ; Высветить исходную
                        ; картинку
        mov  si,OFFSET DstPict ;
        mov  cx,SrcWidth ; Высветить
        mov  dx,00      ; масштабированную
        call PicDisp    ; картинку
        call ClrBuff    ; Очистить буфер
                        ; клавиатуры
NxtCheck:
        mov  ah,01h     ;
        int  16h        ; Буфер пуст?
        jnz  NxtCheck   ; Да-проверить еще раз
        mov  ah,00h     ; Нет - проверить,
                        ; нажата ли
        int  16h        ; клавиша ESC
        cmp  ah,01h     ; ESC нажата?
        jnz  NxtCheck   ; Нет - проверить
                        ; еще раз
        call Exit       ; Да - выйти в DOS
SclTest endp

include SCALE.INC       ; Подпрограмма
                        ; масштабирования

; PicDisp высвечивает на экране картинки
; в формате .PUT
; DS:SI - указывает на картинку
; CX - горизонтальная координата
; DX - вертикальная координата
PicDisp proc near
        mov  ax,0a000h  ;
        mov  es,ax      ; Установить ES
                        ; на начало видеопамяти
        call DefPoz     ; Определить место
                        ; в видеопамяти
        mov  dx,80d     ; dx=числу байтов
                        ; строки экрана
        call BytesPLn   ; Заслать в BX число
                        ; байтов в строке
        sub  dx,bx      ; dx байтов не занято
                        ; картинкой
        mov  cx,[si+02] ; Заслать количество
                        ; строк в картинке
        add  si,04      ; Указатель -
                        ; на начало картинки
        cld
NxtLine:
        push cx         ;
        mov  cx,bx      ; BX - число байтов
                        ; в строке
        rep  movsb      ; Высветить строку
        add  di,dx      ; Перейти к следующей
                        ; строке
        pop  cx         ;
        loop NxtLine    ; Если есть еще
                        ; строки, высветить
        ret             ; Если нет, закончить
PicDisp endp

; BytesPLn определяет число байтов
; в строке изображения
BytesPLn proc near
        mov  bx,[si]    ; Прочитать длину
                        ; строки в пикселах
        add  bx,07
        shr  bx,1       ; Поделить
        shr  bx,1       ; на
        shr  bx,1       ; 8
        ret
BytesPLn endp

; DefPoz определяет место картинки
; в видеопамяти
; На входе: CX - горизонтальная координата
;           картинки
;         DX - вертикальная координата
; На выходе: DI указывает на начало
;           изображения
;         относительно сегмента 0A000
;         Для простоты изображение помещается
;           на границе байта.
DefPoz  proc near
        mov  ax,80d     ; ax=640/8 - число
                        ; байтов в строке
                        ; экрана
        mul  dx         ;
        mov  di,ax      ; di - число байтов
                        ; в пропущенных строках
        add  cx,07      ;
        shr  cx,1       ; cx - число байтов
        shr  cx,1       ; в пропущенных
        shr  cx,1       ; столбцах
        add  di,cx      ; di - смещение
                        ; картинки
                        ; в видеопамяти
        ret
DefPoz  endp

SetVMode proc near
        mov  ah,00
        mov  al,10h     ; Режим EGA 640x350
                        ; 16 цветов
        int  10h
        jmp  SetPlt     ; Установить регистр
                        ; палитры
SetVMode endp

SetPlt  proc near
        mov  ah,10h     ; Установить регистры
                        ; палитры
        mov  al,00      ; Установить один
                        ; регистр
        mov  bl,0fh     ; Номер регистра
        mov  bh,07h     ; Установить белый цвет
        int  10h        ; 00138
        ret
SetPlt  endp

ClrBuff proc near
        mov  ax,00h
        mov  es,ax
        mov  ax,es:[41ah] ; Приравнять указатели
                        ; головы и хвоста
        mov  es:[41ch],ax ; буфера клавиатуры
        ret
ClrBuff endp

Exit    proc near
        mov  ah,0
        mov  al,cs:old_v_mode
        int  10h        ; Восстановить
                        ; видеорежим
        mov  ah,05
        mov  al,cs:old_d_page
        int  10h        ; Восстановить
                        ; видеостраницу
        mov  ah,4ch
        int  21h        ; Выйти в DOS
Exit    endp

old_v_mode db           (?)
        ; Прежний видеорежим
old_d_page db           (?)
        ; Прежняя видеостраница
SrcPict:                ; Начало Src-картинки
SrcWidth  dw            11d
        ; Ширина Src-картинки
SrcHight  dw            11d
        ; Высота Src-картинки
SrcBody   db            00,00,1f,80,60
          db            40,40,00,40,00
          db            3f,00,00,0c0,00
          db            40,60,40,1f,80
          db            00,00
        ; Сама картинка - буква "S"
        ; (См. рис. 1)
DstPict:                ; Начало Dst-картинки
DstWidth  dw            200d
        ; Ширина Dst-картинки
DstHight  dw            156d
        ; Высота Dst-картинки
DstBody:                ; Здесь будет
                        ; масштабированная
                        ; картинка
cseg  ends
      end  SclTest
```

Процессоров стройные ряды...

Д. Арапов

Дайджест книги Джерри Л. Поттера «Ассоциативные вычисления. Парадигма программирования для компьютеров с массовым параллелизмом»

Одно из магистральных направлений развития современной информатики — параллельные вычисления. Выбор той или иной модели параллельных вычислений определяется, с одной стороны, технической реализуемостью адекватного этой модели компьютера, а с другой — набором задач, которые можно эффективно решать в рамках выбранной модели.

Существующие модели можно условно разделить на два больших класса. Первый класс предполагает наличие единого источника команд. Одновременно на всех процессорах выполняется одна и та же команда, но над разными элементами данных. Класс получил название SIMD, что расшифровывается как Single Instruction Multply Data (одна инструкция — множество данных). Другой класс архитектур предполагает наличие на каждом процессоре не только блоков ALU и выборки данных, но и локального счетчика команд и блока их выборки. Называется этот класс MIMD,что представляет собой аббревиатуру слов Multiply Instruction Multiply Data (много инструкций — много данных).

Ко второму классу относятся знаменитые суперкомпьютеры Cray и многочисленные вычислительные системы, базирующиеся на транспьютерах и традиционных микропроцессорах. Теоретически системы MIMD способны достичь более высокой производительности, но имеются обстоятельства, не позволяющие забывать об идеях SIMD: во-первых, процессорный элемент SIMD может быть значительно проще, а, следовательно, меньше по размерам и дешевле, что, в свою очередь, позволяет строить на основе этих элементов системы с массовым параллелизмом; во-вторых, программирование MIMD-машин — дело чрезвычайно трудное и чреватое труднообнаруживаемыми ошибками. (Сомневающиеся в этом отсылаются к сборнику [1]).

Программирование SIMD значительно проще, так как структура программы близка к традиционной последовательной. Основное отличие состоит в том, что каждая элементарная инструкция выполняется на нескольких процессорах одновременно.

Первым компьютером с SIMD-архитектурой был ILLIAC-IV, построенный еще в 1960г. для решения дифференциальных уравнений в частных производных. В настоящее время имеется несколько типов компьютеров, использующих SIMD-параллелизм. К их числу относится и Connection Machine, который в настоящее время считается самым быстрым (10 млрд. операций в секунду). Он содержит 64K процессорных элементов.

Процессорный элемент SIMD можно сделать очень простым, как бы повернув историю развития микропроцессоров вспять. Вспомним, что первым микропроцессором был i4004, оперировавший словами длиной в 4 бита. Со временем длина слова возрастала, появилась возможность при помощи всего одной команды складывать (вычитать, умножать и так далее) 8-, 16-, 32-битовые слова. Это безусловно повысило скорость вычислений, но процессор стал гораздо сложнее. Создатели многих SIMD-архитектур решили вернуться к истокам и заставили процессор работать только с предельно короткими данными, имеющими длину всего один бит. Процессор можно сделать очень простым, компактным и дешевым, что позволит поместить в машину очень много данных элементов. Такая примитивная архитектура процессорного элемента называется побитовой (bit-serial).

Этот простой процессор объединяется в одно устройство с блоком памяти длиной в слово сначала логически, а в будущем и конструкционно. В отличие от традиционных архитектур, где процессор и память обмениваются целыми словами и их длина обычно не превышает 32 битов, в bit-serial SIMD-компьютерах используют слова длиной до 64 Кбитов.

Дж. Поттер предложил развитие традиционной SIMD-архитектуры. Компьютер включает устройство управления, связанное через шину с множеством вычислительных элементов, каждый из которых представляет собой процессорный элемент, объединенный с достаточно длинным словом памяти. Программа может некоторым образом разделить слово памяти на логически связанные группы битов — поля. При этом однобитовые поля могут рассматриваться как булевские флаги, а поля, состоящие из многих битов, моделируют числа. Свою модель он назвал SITDAC. Первые две буквы аббревиатуры нам уже знакомы — Single Instruction. Модель подразумевает, что каждый такт устройство управления выдает команду, которая выполняется процессорными элементами. Следующие две буквы соответствуют словам Tabular Data. В рамках модели удобно рассматривать компьютер как таблицу, каждая строка которой — вычислительный элемент, а колонки соответствует логическим полям. Последние две буквы (AC) происходят от слов Associative Computing.

Типичный участок вычислений в модели SITDAC состоит из трех фаз: поиска, выполнения и извлечения результатов.

Фаза поиска определяет те вычислительные элементы, которые будут активны на последующем участке вычислений. Устройство управления выдает булевское условие, которое параллельно проверяется на каждом из процессоров. Те процессоры, для которых предикат истинен, помечаются как активные.

Следующая фаза состоит в параллельном выполнении всеми активными процессорами последовательности команд, выдаваемых устройством управления. При

J.L. Potter. Associative Computing. A Programming Paradigm for Massively Parallel Computers.– NY: Plenum Press, 1992.

этом каждый процессорный элемент выполняет команду над своими локальными данными.

На последней фазе результаты вычислений попадают в устройство управления. Поскольку в общем случае результатов много (столько же, сколько активных вычислительных элементов на данном участке), эта фаза выполняется последовательно, причем порядок выдачи результатов разными активными элементами не определен.

Моделью ассоциативных вычислений может служить школьный класс, где каждый ученик соответствует вычислительному элементу, а учитель — устройству управления. На фазе поиска учитель говорит: «Все, кто выполнил задание, поднимите руки». На второй фазе он обращается к поднявшим руки: «Возьмите лист с решением задачи». На третьей фазе он просит показать решение. При этом он должен работать индивидуально с каждым из учеников. Ученики подходят к учителю по одному в произвольном порядке и показывают, как они выполнили задание.

Последовательные программы удобно писать на языках Fortran, C, Pascal... Для записи MIMD-программ удобны Occam и Ada. Для записи ассоциативных вычислений разработан специальный язык ASC.

В этом языке отсутствуют адреса и указатели. (Заметим, что многие специалисты по языкам программирования считают, что указатели столь же вредны для структур данных, сколь оператор GOTO для кода.) При ассоциативных вычислениях их заменяют флаги, отмечающие вычислительные элементы.

ASC-программы имеют дело со скалярными и параллельными элементами данных, которые соответствуют переменным и массивам в традиционных языках программирования. Индекс $ указывает на все активные элементы. Для работы с отдельными элементами массива (извлечения) служат индекс-параллельные переменные. В бинарных выражениях могут смешиваться скалярные и параллельные элементы. Если хотя бы один из аргументов — параллельный, то и результат тоже параллельный. Полагают, что в этих случаях скалярный элемент размножается в необходимом количестве экземпляров.

Рассмотрим наиболее важные операторы этого языка. **Ассоциативный оператор if.** Если логическое выражение представляет собой скаляр, то выполняется традиционный оператор if. Если логическое выражение дает параллельный результат, то выполняет следующую последовательность действий: поиск по условию, выполнение then части, поиск дополнения, выполнение else части.

```
if ученик_выполнил_задание[$] then
оценка_ученика[$] = 5;
else оценка_ученика[$] = 2;
endif
```

Отдельный интерес может представлять случай, когда поиск по условию не дал ни одного активного элемента. Для программирования реакции в таких случаях служит **оператор if-elsenany:**

```
if ученик_выполнил_задание[$] then
оценка_ученика[$] = 5;
elsenany учитель_повторяет_материал;
endif
```

Таким образом if реализует параллельный цикл. В языке предусмотрен и другой тип цикла — итеративный. Его синтаксис несколько сложнее:

```
for индекс in параллельное условное
выражение
Последовательность действий
[elsenany
Последовательность действий]
endfor
```

С помощью итеративного цикла можно описать последовательность опроса учеников, выполнивших задание.

Передача данных от одного процессора к другому. Два типа передачи данных

Ключевым моментом, определяющим эффективность многопроцессорных вычислительных систем, является способ передачи данных от процессора к процессору.

Модель SITDAC предполагает 2 режима передачи данных от одного вычислительного элемента к другому. Первый режим состоит в глобальном прямом доступе. Данные из элемента-источника извлекаются в устройство управления, после чего рассылаются всем элементам, установившим флаг приема. Несмотря на свою универсальность и нетребовательность к аппаратуре, этот способ не может считаться удовлетворительным в случае, когда одновременно много элементов нуждаются в передаче сообщений от одного к одному. Возможен и другой режим — синхронная передача соседу. Предполагается, что элементы объединены в сеть, по которой по соответствующей команде от устройства управления данные передаются от элемента к его левому (правому) соседу. Сеть может быть одномерной (у каждого элемента 2 соседа), двумерной, трехмерной и так далее.

При наличии сети соответствующей размерности чрезвычайно упрощается, например, решение сеточных задач уравнений математической физики. Реализация сети, однако, усложняет аппаратуру. Кроме того, в программах придется учитывать размерность сети и ее геометрию, что плохо согласуется с другими аспектами ассоциативного программирования, которое по своей природе машинно-независимо.

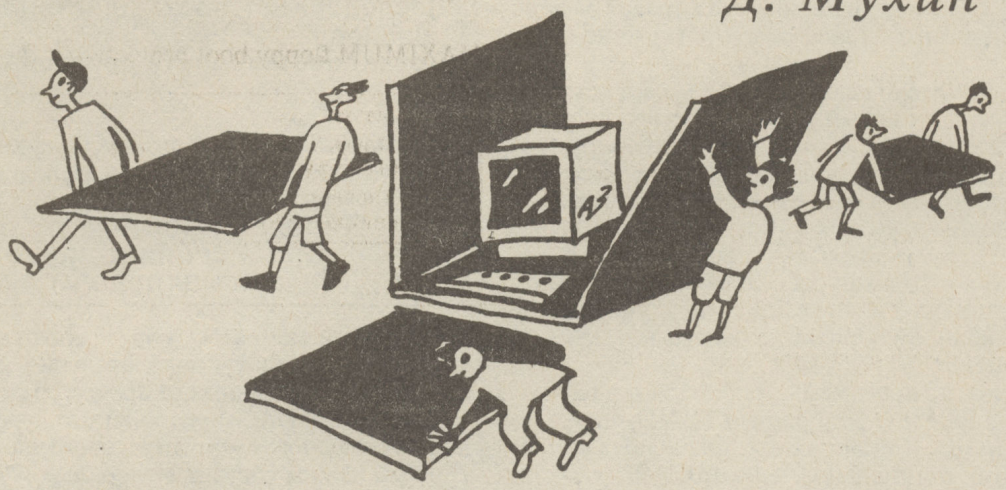
Дж. Поттер в своей книге приводит реализацию на языке ASC интерпретатора языка Prolog. Я думаю, что трудно переоценить значение использования массового параллелизма для логического вывода. Совершенно очевидно, что ассоциативные компьютеры и модель вычислений SIDTAC могли бы найти применение в тех областях, где требуется массовое геометрическое распараллеливание: для построения фотореалистических изображений и мультипликации, для распознавания образов и для моделирования физических процессов на уровне молекул. Наличие ясного структурного языка ASC подтверждает зрелость этой модели. И несмотря на то, что широкое применение ассоциативных вычислений не является делом ближайшего будущего, следует помнить, что нет ничего практичнее хорошей теории. ■

Литература

1. Программирование на параллельных вычислительных системах/ Под ред. Р. Бэбба II.– М.: Мир, 1991.

Удаление защиты PC–VAULT

Д. Мухин



Во многих студенческих ВЦ на компьютерах установлены всевозможные системы защиты. За счет этого многие студенты не могут пробиться к и без того немногочисленным компьютерам, которые при этом иногда бывают свободны. Таким образом обеспечены только «особы, приближенные к императору» [1]. Можно ли помочь остальным?

Я попытаюсь помочь тем, у кого винчестеры защищены программой PC-VAULT. При работе с этой системой мне очень помогла статья [2]. Она побудила меня к просмотру загрузочных секторов различных винчестеров, в том числе и винчестера, защищенного системой PC-VAULT. Дисассемблируя этот сектор, я обнаружил, что система PC-VAULT

1) шифрует таблицу разделов (Partition Table);

2) изменяет подписи загрузочных секторов всех разделов винчестера (в том числе и неактивных) с AA55h на 4161h.

Код дешифратора, который одновременно является и шифратором, находится в загрузочном секторе винчестера (смещение 59h). Его текст представлен в листинге 1.

Дешифратор оформлен в виде процедуры. При этом предполагается, что в SI содержится смещение таблицы разделов (1BEh). Разность 1BEh - 1B8h = 06h является смещением ключа, с помощью которого шифруется-дешифруется таблица разделов. Этот ключ устанавливается во время установки системы на винчестер и сохраняется неизменным.

В результате проверки

```
CMP   BYTE PTR [SI-01BEh], 37h
```

флаг нуля не устанавливается, так как байт со смещением 0 в таблице разделов содержит 9Fh. Если кто-то попробует дисассемблировать загрузочный сектор винчестера, на котором установлена система PC-VAULT, то заметит в коде множество непонятных ветвлений и переходов, а также частей кода, которые никогда не выполняются. Это, по-видимому, сделано для осложнения «вскрытия» алгоритма работы системы. С этой же целью вставлена и проверка в начале процедуры шифратора-дешифратора (дело в том, что четное количество обращений к этой процедуре не изменяет участок памяти, для которого она вызывалась). Интересно, что имеется еще одна копия шифратора, расположенная в другом секторе. Она-то и используется. Кажется странным, что такой важный фрагмент кода был продублирован «для отвода глаз».

Обладая приведенной выше информацией, можно написать программу, которая снимает с винчестера систему PC-VAULT. Для этого необходимо:

1) записать в загрузочный сектор винчестера код загрузки и выполнения корневого сектора активного раздела;

2) записать туда же (со смещением 1BEh) расшифрованную таблицу разделов;

3) восстановить подпись загрузочного сектора активного раздела.

Программа, выполняющая эти действия, представлена в листинге 2. Для снятия защиты с винчестера нужно загрузиться с дискеты и запустить программу. После этого Вы получите свободный доступ к винчестеру.

Для окончательного удаления следов системы необходимо:

1) восстановить подписи загрузочных секторов неактивных разделов;
2) удалить файлы CLEANDSK.DRV, MEM-STAT.SYS;
3) снять атрибут ReadOnly с файлов CONFIG.SYS и AUTOEXEC.BAT;
4) удалить строку

```
DEVICE=CLEANDSK.DRV
```

из файла CONFIG.SYS.

Эти действия можно проделать вручную или программным путем. Надеюсь, что эта информация облегчит нелегкую жизнь некоторых программистов.

Примечание. Материал составлен на основе версии 4.2. При работе с программой, приведенной в листинге 2, рекомендуется сохранить таблицу разделов Вашего диска (например, с помощью программы DiskEdit из пакета Norton Utilities). Программа (листинг 2) имеет ограничение. Не следует ей пользоваться при установленной опции Select OPTIONS/ MAXIMUM floppy boot protection. ■

Литература

1. И. Ильф, Е. Петров. 12 стульев.– М.: Художественная литература, 1992.
2. В. Аношин. Управление загрузкой ОС на IBM PC// Монитор.– 1991.– № 1.

Листинг 1.
Код шифратора–дешифратора

```
        MOV   AX, [SI-01B8H]; Ключ
        CMP   BYTE PTR [SI-01BEH], 37H
        JE    M2
        MOV   BYTE PTR [SI-01BEH], 37H
        MOV   CX, 20H         ; Размер таблицы
                              ; разделов в словах
M1:
        ROL   AX, 1
        ROL   AX, 1
        ROL   AX, 1
        ADD   AX, 5CA3H
        XOR   [SI], AX
        INC   SI
        INC   SI
        LOOP  M1
M2:
        RET                                   ■
```

Листинг 2.
Программа удаления PC–VAULT version 4.2

```
TITLE   Программа удаления PC-VAULT version 4.2
;
; ************ Выдать сообщение ************ ;
;
WR_MES  MACRO MES, LEN
        LEA   DX, MES
        MOV   CX, LEN
        MOV   AX, 4000H
        MOV   BX, 0002H
        INT   21H
ENDM
;
; ***************************************** ;
;
CSEG    SEGMENT 'Code'
ASSUME  CS:CSEG, DS:CSEG, SS:CSEG, ES:CSEG
        ORG   100H
BEGIN   PROC
;
; Считать таблицу разделов
;
        MOV   DX, 0080H   ; HDD, сторона 0
        MOV   CX, 0001H   ; Цилиндр 0, сектор 1
        LEA   BX, OLD_SEC
        MOV   SI, 0201H   ; Считать 1 сектор
        CALL  RW
        JNC   M1 ; Если нет ошибки
        WR_MES  ERR1, LEN1  ; Выдать сообщение
        JMP   TERM ; Закончить
;
; Проверить наличие защиты
;
M1:
        LEA   SI, OLD_SEC
        CMP   WORD PTR [SI], 0EB9FH
        JE    M01           ; Защита есть
        WR_MES  INF, LEN4
        JMP   TERM          ; Закончить
;
; Расшифровать таблицу разделов
;
M01:
        LEA   SI, OLD_SEC
        ADD   SI, 01BEH     ; Адрес старой
                            ; таблицы разделов
        LEA   DI, OLD_SEC
        ADD   DI, 0059H
        MOV   DECOD, DI     ; Адрес дешифратора
        CALL  WORD PTR DECOD
;
; Записать таблицу разделов
; в область нового загрузчика
;
        CLD
        LEA   SI, OLD_SEC
        ADD   SI, 01BEH
        LEA   DI, PAT_TAB
        MOV   CX, 0020H
        REP   MOVSW     ; Переписать дешифрованную
                        ; таблицу
;
; Записать новый загрузчик
;
        MOV   DX, 0080H   ; HDD, сторона 0
        MOV   CX, 0001H   ; Цилиндр 0, сектор 1
        LEA   BX, NEW_SEC
        MOV   SI, 0301H   ; Записать 1 сектор
        CALL  RW
        JNC   M2          ; Если нет ошибки
        WR_MES  ERR2, LEN2  ; Выдать сообщение
        JMP   TERM        ; Закончить
;
; Найти активный раздел
;
M2:
        LEA   SI, PAT_TAB   ; Начало
                            ; таблицы разделов
M3:
        CMP   BYTE PTR [SI], 80H
        JE    M4            ; Переход, если активный
        ADD   SI, 0010H   ; Следующий раздел
        JMP   M3
;
; Считать загрузочный сектор
; активного раздела
;
M4:
        MOV   DX, [SI]        ; Диск/головка
        MOV   CX, [SI+2]      ; Дорожка/сектор
        LEA   BX, OLD_SEC
        MOV   SI, 0201H       ; Читать 1 сектор
        PUSH  CX
        PUSH  DX              ; Запомнить размещение
```

```
        CALL   RW
        JNC    M5
        WR_MES  ERR1, LEN1  ; Выдать сообщение
        JMP    TERM         ; Закончить
;
; Восстановить подпись
;
M5:
        LEA    SI, OLD_SEC
        ADD    SI, 01FEH    ; Адрес подписи
        MOV    WORD PTR [SI], 0AA55H
;
; Записать восстановленный сектор
;
        POP    DX
        POP    CX
        LEA    BX, OLD_SEC
        MOV    SI, 0301H    ; Записать восстанов-
                            ; ленный сектор
        CALL   RW
        JNC    TERM
        WR_MES  ERR2, LEN2  ; Выдать сообщение
;
; Закончить программу
;
TERM:
        MOV    AX, 4C00H    ; Окончить программу
        INT    21H
BEGIN   ENDP


;
; ********* Считать/записать сектор ******* ;
;
RW      PROC
        MOV    DI, 0005H    ; Пять попыток
MR1:
        MOV    AX, SI       ; Операция
        INT    13H
        JNC    MR2          ; Переход, если нет ошибки
        XOR    AX, AX
        INT    13H          ; Сброс дисковода
        DEC    DI           ; Следующая попытка
        JNE    MR1
        STC                 ; Ошибка
MR2:
        RET
RW      ENDP
;
; ****************************************** ;
; **                                      ** ;
; **              Код загрузчика          ** ;
; **                                      ** ;
; ****************************************** ;
;
        ORG    300H
NEW_SEC LABEL  BYTE
        CLI                 ; Запрет прерываний
        XOR    AX, AX       ; Обнулить AX
        MOV    SS, AX       ; SS = 0
        MOV    SP, 7C00H    ; Вершина стека
                            ; на начало кода
        MOV    SI, SP       ; В SI - адрес
                            ; начала кода
        PUSH   AX
        POP    ES           ; ES = 0
        PUSH   AX
        POP    DS           ; DS = 0
        STI                 ; Разрешить прерывания
        CLD                 ; Обработка строк вперед
        MOV    DI, 0600H    ; Новое место загрузчика
        MOV    CX, 0100H    ; Переписать 512 байтов
        REPNZ MOVSW
        DB     0EAH
        DW     061DH, 0000H ; Переход на Z1
Z1:
        MOV    SI, 07BEH    ; Начало таблицы
                            ; разделов
        MOV    BL, 04H      ; 4 раздела
Z01:
        CMP    BYTE PTR [SI], 80H
        JE     Z2           ; Переход, если активный
        CMP    BYTE PTR [SI], 00H
        JNE    Z3 ; Переход, если ошибка в таблице
        ADD    SI, 0010H    ; Следующий раздел
        DEC    BL
        JNE    Z01
        INT    18H          ; Бейсик
Z2:
        MOV    DX, [SI]     ; Диск/головка
        MOV    CX, [SI+2]   ; Дорожка / сектор
        MOV    BP, SI   ; Запомнить адрес раздела
Z02:
        ADD    SI, 0010H    ; Проверить остаток
                            ; таблицы
        DEC    BL
        JE     Z4 ; Переход, если таблица правильная
        CMP    BYTE PTR [SI], 00H
        JE     Z02
Z3:
        MOV    SI, 068BH    ; MES1
Z03:
        LODSB
        CMP    AL, 00H
        JE     Z5           ; Конец строки?
        PUSH   SI
        MOV    BX, 0007H    ; Черный и белый
        MOV    AH, 0EH      ; Вывод символа
        INT    10H
        POP    SI
        JMP    Z03          ; Следующий символ
Z5:
        JMP    Z5           ; Зациклить
Z4:
        MOV    DI, 0005H    ; 5 попыток
Z04:
        MOV    BX, 7C00H
        MOV    AX, 0201H    ; Читать 1 сектор
        PUSH   DI
        INT    13H
        POP    DI
        JNB    Z6           ; Нет ошибок
        XOR    AX, AX       ; Сброс дисковода
        INT    13H
        DEC    DI
        JNE    Z04
        MOV    SI, 06A3H    ; MES2
        JMP    Z03          ; Выдать сообщение
Z6:
        MOV    SI, 06C2H    ; MES3
        MOV    DI, 7DFEH
        CMP    WORD PTR [DI], 0AA55H
        JNE    Z03          ; Неверная подпись
        MOV    SI, BP       ; Востановить адрес
                            ; начала раздела
        DB     0EAH         ; Передать управление
        DW     7C00H, 0000H ; загрузчику
MES1    DB     'Invalid partition table', 0
MES2    DB     'Error loading operating system', 0
MES3    DB     'Missing operating system', 0
        ORG    4BEH
PAT_TAB DB     40H DUP(0)
        DB     55H, 0AAH    ; Подпись
;
; ****************************************** ;
;
OLD_SEC DB     512 DUP( 0 )
ERR1    DB     0DH, 0AH
        DB     'Error reading partition table.'
LEN1    EQU    $ - ERR1
ERR2    DB     0DH, 0AH, 'Error writing sector.'
LEN2    EQU    $ - ERR2
INF     DB     0DH, 0AH, 'There is no protection'
        DB     ' on your HDD.'
LEN4    EQU    $ - INF
DECOD   DW     0
;
CSEG    ENDS
        END    BEGIN
```

Если у Вас проблемы с принтером СМ6337

С. Дорошев

Не гонялась бы, ты, кума, за дешевизной!

Народная мудрость

Принтеры СМ6337, которые выпускаются уже не один год, привлекают ту часть потребителей, для которых определяющим фактором является невысокая цена. К тому же общие характеристики этих устройств позволяют использовать их в некоторых профессиональных приложениях. Принтер широкий, имеет относительно высокую скорость печати и достаточно полный, соответствующий фактическому стандарту набор команд управления.

Недавно автору пришлось близко познакомиться с устройством этого типа. Устранение «персональных особенностей» конкретного экземпляра отняло массу времени. Возможно, индивид попался уникальный, но очевидно, что эти «особенности» принтер получил в процессе сборки.

Следует добавить, что поначалу трудно было выделить механические дефекты на фоне недостатков встроенной программы управления. Часто из-за нелепой реакции принтера на подаваемые ему коды под вопросом была исправность электрической схемы, что также затрудняло и затягивало разбирательство. Потребовалось найти несколько экземпляров устройств этого типа и сравнить их работу. Стало ясно, что если первый опыт общения с принтером СМ6337 удачный, то не стоит обольщаться. Это устройство всегда готово преподнести «сюрприз». О своих «достоинствах» оно начинает заявлять сразу, как только его задача становится отличной от совсем элементарной.

Когда, выловив огромное количество «подарков» производства, начинаешь постигать «логику» работы принтера СМ6337, не перестаешь удивляться, задавая автору (или авторам) программы управления риторический вопрос: «Почему?». Почему нельзя было предусмотреть обработку ситуации переполнения буферной памяти принтера? Почему комбинация кодов ВК ПС ВК (0Dh, 0Ah, 0Dh) выводит устройство из себя так, что после этого ему лучше всего помогает отключение питания? Почему режим NLQ и печать графической информации конфликтуют между собой?... То, что печать нельзя просто перевести в альтернативный код переключателем, на фоне таких безобразий кажется совсем несущественным неудобством.

Ради возможности использования печатающего устройства СМ6337 при выводе информации пришлось заняться написанием драйвера для этого принтера. Результат оказался вполне удовлетворительным.

В версии 2.2 драйвер СМ6337.EXE обрабатывает следующие ситуации:

1. Отключает режимы NLQ и двойного удара перед выводом на печать графической информации и восстанавливает их по завершении процедуры (если они были включены).
2. Следит за тем, чтобы буфер принтера не переполнялся. Если объем графических данных больше заданного количества байтов, то драйвер «разрезает» посылаемую на принтер графическую информацию на части.
3. Перед выводом кода «возврат каретки» (0Dh) драйвер выдает на принтер пробел, при этом «сумасшествие» принтера по поводу последовательности 0Dh, 0Ah, 0Dh устраняется.
4. Временно отключает режим NLQ, если в выводимой последовательности встречается псевдографический символ, так как принтер в режиме NLQ псевдографики не имеет.
5. Задает паузу 0,01 секунды между выводимыми символами, иначе в режиме TURBO на компьютере AT/286 (10 МГц) входные цепи принтера начинают давать сбои. Пауза никак не сказывается на скорости печати принтера.
6. При прохождении команды установки в исходное состояние и через каждые 256 посылаемых в принтер символов драйвер отслеживает включение альтернативной кодовой таблицы (если не задан режим работы в основной таблице).
7. В любых режимах драйвер не искажает графические данные.
8. Драйвер имеет режим (параметр в строке вызова: K1), при котором он по определенной логике заменяет команду подачи бумаги (Esc J ...) на 1/216 дюйма командой подачи на 2/216 дюйма. Этот режим пришлось ввести из-за того, что принтер почему-то игнорирует команду подачи на 1/216 дюйма.

Драйвер для принтеров OKI Data Microline создавался уже на базе драйвера СМ6337. Задача в этом случае была очень простой. Она состояла в том, чтобы эмулировать недостающие команды вывода графической информации. При работе этого драйвера заметных искажений графического изображения из-за дробного коэффициента при преобразовании растра не происходит, а работать с принтером становится проще. Эффект от применения драйвера значительный, так как он снимает проблему неполной совместимости OKI с современными принтерами Epson.

Думаю, что описанные в статье проблемы коснулись многих владельцев IBM PC совместимых компьютеров.

Распространенность относительно недорогих принтеров велика, поэтому редакция журнала «Монитор» и автор готовы оказать помощь в приобретении упомянутых здесь драйверов. Мы надеемся, что информация, приведенная в этой публикации, уменьшит количество проблем, сваливающихся на головы отечественных пользователей компьютерной техники. ■

EPSON

японская сборка

БАНКОВСКИЕ МАТРИЧНЫЕ

Быстродействующие принтеры, специально созданные для работы с большими объемами информации, печатают только на перфорированной бумаге, автоматически распознают форматы.

DFX-8000.....$ 2800 - 3350;
DFX-5000.....$ 1750 - 2450;

ЛАЗЕРНЫЕ

EPL 5200.....$ 830-930.

ОФИСНЫЕ МАТРИЧНЫЕ

LX-100.....$ 205 - 248;
LQ-100.....$ 270 - 308;

Удобная форма, легкий вес, автоподача до 50 листов отличают эти модели.

FX-1170.....$ 480 - 640;
LQ-1070.....$ 540 - 650;
LX-1050.....$ 290 - 380;

Широкие принтеры формата А3; самые известные в своем классе.

Гарантия 12 месяцев

КОМПЬЮТЕРЫ ФИРМЫ IBM: PS/1, PS/VP, PS/2, AS/400 -любые заказные конфигурации.

ДИСКЕТЫ 3M- чья долговечность сравнима с суммой лет всех сотрудников вашей фирмы, пожизненная гарантия, гарантируем восстановление информации с дискет, не читаемой обычным образом; дискеты - 3,5; 5,25; картриджи к стримерам - 120 Мб, 150 Мб,250 Мб, 525 Мб.

БЫТОВАЯ ЭЛЕКТРОНИКА фирм: SONY, TOSHIBA, DAEWOO, широкий ассортимент товаров MOULINEX.

Существуют ли шедевры, приносящие ежедневную пользу?

-Да, их поставляет компания "Радом"

Novell NetWare

Бесплатно даются Norton Backup и Norton AntiVirus только с русскими версиями Novell NetWare 3.11

	Цена фирмы	Наша цена
10 пользователей		941 $
20 пользователей		1316 $
50 пользователей	~~2495 $~~	2121 $
100 пользователей	~~3495 $~~	2621 $

IBM

OS/2 2.1	~~340 $~~	287 $
Extended Serviceswith Data Base	~~3066 $~~	2759 $
Lan Server 3.0	~~1243 $~~	1049 $

и другие программные продукты IBM

AutoDESK

AutoCAD R12+AME 2	~~1824 $~~	1632 $
AutoCAD R12	~~1596 $~~	1428 $
AutoCAD R11 386+AME 1.0	~~1140 $~~	1040 $
AutoCAD R11286	~~1026 $~~	901$

Lotus

Помощник делового человека Lotus Organizer Английский вариант	~~149 $~~	99 $
Русифицированный вариант		75 $
Электронная таблица 1-2-3 v 4.0 For Windows	~~495 $~~	99 $
Текстовый процессор (издательская система) Ami Pro v 3.0 For Windows	~~495 $~~	99 $
Пакет деловой графики Freelance v 2.0 For Windows	~~495 $~~	99 $

Торговые залы в Москве:

"Перспективные технологии",
Звенигородское шоссе, д. 9,
(095) 256-40-30;
маг. "Москва", ул. Тверская, д.8,
тел.: (095) 229-84-81;
маг. "Дом книги". ул. Новый Арбат, д. 9.

ЕКАТЕРИНБУРГ
"Радом - Восток",
Тел.: (3432) 22-52-08

ТАГАНРОГ
"Радом-Т",
Тел.: (86344) 6-22-15

Телефон для оптовых покупателей: (095) 256-11-64

Телефон для справок по всем залам: (095) 256-44-73, 256-40-30. Факс: (095) 259-27-27.

© 1993 Радом®

Радом Р. А.

InVecom '93

Санкт–Петербург 7–11 декабря 1993 г. Выставочный комплекс в Гаване

КРУПНЕЙШАЯ КОМПЬЮТЕРНАЯ ВЫСТАВКА СЕВЕРО–ЗАПАДА

ТЕМАТИКА

— программные средства и системы;
— вычислительные средства и системы;
— техника связи и система коммуникаций;
— техника и оборудование для офиса;
— обучающие технологии и системы.

ОРГКОМИТЕТ:
191040, СПб, а/я 19,
"Инвеком"
Тел.: (812) 112–2040
Факс: (812) 112–2348

НЕОГРАНИЧЕННЫЕ ВОЗМОЖНОСТИ НА ОГРАНИЧЕННОЙ ПЛОЩАДИ

Монитор

5302

Ваша программа, на Вашем компьютере, но гораздо быстрее!

Работу с этими и другими программами ускоряют сопроцессоры Intel

Математические сопроцессоры фирмы Intel:

— **привлекают крупнейших разработчиков программного обеспечения.** Свыше 600 таких компаний во всем мире ориентируются на математические сопроцессоры фирмы Intel, и уже свыше 2100 прикладных программ активно их используют;

— **предлагают проверенное решение для любого компьютера от XT до i486 SX.** Математические сопроцессоры фирмы Intel являются индустриальным стандартом. Во всем мире буквально миллионы пользователей с удовлетворением пожинают плоды повышенной производительности;

— **обеспечивают совместимость.** Поскольку сопроцессоры фирмы Intel разработаны специально для работы совместно с микропроцессорми Intel, на основе которых созданы все персональные компьютеры, они обеспечивают 100%–ную совместимость;

— **означают качество фирмы Intel.** Математические сопроцессоры, изготовленные фирмой Intel, являются наивысшим стандартом в мире и имеют пятилетнюю гарантию;

— **легко устанавливаются.** Надо лишь вставить сопроцессор в отведенное для него гнездо – и готово!

intel®